Model-Based Diagnostic Decision-Support System for Satellites

Alexander Feldman
University College Cork
Cork, Ireland
a.feldman@ucc.ie

Helena Vicente de Castro
CGI
Rotterdam, The Netherlands
helena.vicente.de.castro@cgi.com

Arjan van Gemund
Delft University of Technology
Delft, The Netherlands
a.j.c.vangemund@tudelft.nl

Gregory Provan
University College Cork
Cork, Ireland
g.provan@cs.ucc.ie

***Abstract*—We propose a novel framework for Model-Based Diagnosis (MBD) that uses active testing to decrease the diagnostic uncertainty. This framework is called LYDIA-NG and combines several diagnostic, simulation, and active-testing algorithms. We have illustrated the workings of LYDIA-NG by building a LYDIA-NG-based decision support system for the Gravity field and steady-state Ocean Circulation Explorer (GOCE) satellite. This paper discusses a model of the GOCE Electrical Power System (EPS), the algorithms for diagnosis and disambiguation, and the experiments performed with a number of diagnostic scenarios. Our experiments produced no false positive scenarios, no false negative scenarios, the average number of classification errors per scenario is 1.25, and the fault detection time is equal to the computation time. We have further computed an average fault uncertainty of 2.06×10^{-3} which can be automatically reduced to 9.5×10^{-4} by sending a single, automatically computed, telecommand, thus dramatically reducing the fault isolation time.**

TABLE OF CONTENTS

1. INTRODUCTION

This paper reports on the results of an ESA ITI project, called Ground-basEd diagNostIc sUpport System (GENIUS). The main goal of GENIUS is to demonstrate the advantages of adopting Model-Based Diagnosis (MBD) and active testing for decision support in the operational control of satellites. As a target subsystem, we have used the power generation and distribution network of the Gravity field and steady-state Ocean Circulation Explorer (GOCE) satellite. To the best of our knowledge, GENIUS is the first system to apply a model-based active testing algorithm to a real-world system. The results show that MBD and active testing can increase the diagnostic accuracy and decrease the fault isolation time.

For the diagnosis and the active testing of GOCE, we have used the LYDIA-NG modeling language and the LYDIA-NG suite of algorithms. LYDIA-NG can combine the computations of multiple simulation engines. An example of a simulation engine implemented in LYDIA-NG is the analogue electronic simulator which is based on the well-known Simulation Program with Integrated Circuit Emphasis (SPICE). LYDIA-NG also implements several strategies for the generation of fault candidates and a number of algorithms for active testing. These algorithms are based on AI search and include best-first, and bottom-up greedy search. We discuss all these algorithms and their application to the GOCE satellite.

We validate our approach using nine fault scenarios that summarize a class of failures. These scenarios include single- and multiple-fault injections (up to quadruple faults), masking faults, and faults that cannot be disambiguated by design. We have considered a class of intermittent faults as well.

To independently assess the capabilities of the diagnostic and disambiguation tools, we have created the sensor data for the diagnostic scenarios by implementing a model of the GOCE EPS in the ESA-ESOC SIMSAT infrastructure and not the built-in LYDIA-NG simulation suite. All experiments are performed by using two models of the GOCE Electrical Power System (EPS), one used for the generation of telemetry, and the other used by LYDIA-NG for the tasks of MBD and active testing.

The performance of the diagnostic algorithms has been measured in terms of diagnostic metrics. In our experiments we have no false positive scenarios, no false negative scenarios, the average number of classification errors per scenario is 1.25, and the fault detection time is equal to the computation time. We have further computed an average fault uncertainty of 2.06×10^{-3} which can be automatically reduced to 9.5×10^{-4} by sending a single, automatically computed, telecommand, thus dramatically reducing the fault isolation time.

To facilitate integrating LYDIA-NG with real satellite monitoring systems, we have designed an Application Programming Interface (API) for connecting LYDIA-NG to the GOCE ground telemetry and telecommands infrastructure. This allows the automatic translation of telemetry units and the integration of LYDIA-NG in the satellite ground control system.

The contributions of this paper are as follows:

978-1-4577-0557-1/12/$26.00
[1] IEEEAC Paper #2801, Version 4, Updated 29/10/2012.

• We propose a new framework for MBD called LYDIA-NG. LYDIA-NG accommodates algorithms for simulation, diagnosis, and active testing. We measure the diagnostic and disambiguation performance of LYDIA-NG with a number of diagnostic metric. LYDIA-NG provides a way to optimize one or more of these metrics given the specific user environment.
• We build a decision support system on top of LYDIA-NG. This decision support system is used for diagnosing the EPS of the GOCE satellite.
• We achieve decrease in the fault isolation time of EPS failures by applying active testing.
• We introduce a number of EPS diagnostic scenarios that, in addition to computing performance metrics for our approach, can be also used for comparison to other methods.
• We discuss issues related to the interface of LYDIA-NG with a simulator of telemetry data and a SCADA system for control of satellites.

2. RELATED WORK

LYDIA-NG belongs to a class of MBD methods that use continuous-valued models and sensor data, and use entropy-based methods for test selection to disambiguate diagnoses. It is a generalization of LYDIA, which used discrete-value models.

In terms of diagnostics solvers, LYDIA-NG is related to the HyDE (Hybrid Diagnosis Engine) solver [1]. The HyDE-S variant accepts as input interval-valued hybrid models and continuous-valued sensor data. Another solver, FACT [2], can also use continuous-valued models and sensor data, but requires that the model be represented as a hybrid bond graph. Given an anomaly, FACT first uses an observer-based approach (adopted from the FDI community) with statistical techniques for robust fault detection. Fault isolation is performed using qualitative inference, i.e., by matching qualitative deviations caused by fault transients to those predicted by the model.

There is a large body of work that performs test selection during diagnostics inference. The majority of these approaches require discrete-valued Bayesian models and discrete-valued sensor data. For example, Zheng et al. [3] adopt Bayesian network (BN) models, and interleave diagnostics inference with test selection, which is performed in a greedy manner. Test selection uses loopy belief propagation as the inference method, in order to simultaneously compute approximations of marginal and conditional entropies on multiple subsets of nodes in the BN.

In a similar vein, Wu et al. [4] also use BN methods for test selection, applying their approach to computer networks. Quiao et al. [5] also propose BN test selection methods, but reduce inference complexity by identifying the approximate conditional independence of probes, based on leveraging the conditional independence structure of the BN.

Bellala et al. [6] consider a related framework, except that they allow the presence of noise in the model and the observations. Tolpin and Shimony [7] also frame test selection within a probabilistic model, but propose the use of value of information (VOI) as the criterion for selecting the best test.

In contrast to these approaches, LYDIA-NG does not force the use of a discrete-valued probabilistic model, but can make use of the continuous-valued models widely used in many application domains, such as circuit or energy models.

GENIUS, the design of LYDIA-NG and this paper have been influenced by the International Diagnostic Competition (DXC) [8]. The first years of DXC targeted the Electrical Power System (EPS) testbed in the ADAPT lab at NASA Ames Research Center [9] and we have chosen a real-world satellite subsystem that resembles ADAPT.

In the experiments of this paper we have used a number of performance metrics with which we have measured the performance of our diagnostic and disambiguation algorithms (see Sec. 6). These metrics are continuation of the work done for a DXC paper [10] and we propose that future DXC competitions use the up-to-date isolation accuracy and classification error metrics. These metrics assume that the diagnosis is presented as a probabilistic assignment, while in DXC the diagnosis is presented as a set of weighted non-probabilistic assignments. This version simplifies the understanding and the computation of the performance metrics. Further, the uncertainty metric is new. This paper shares with [10] several citations and a small amount of text in the beginning of Sec. 6 (introductory part describing the evolution of the metrics). This is done for readability and being self-contained.

3. CONCEPTS AND DEFINITIONS

In this section we introduce our basic definitions, illustrated on a small artificial electrical circuit that exposes some properties that are present in a real-world power-distribution network of a satellite.

Running Example

All concepts and algorithms in this paper are illustrated with the help of the circuit shown in Fig. 1. This circuit is best considered as hybrid because there are analogue components such as resistors as well as switches (the latter can, of course, be modeled as non-linear analog electronic components but that would unnecessarily increase the modeling complexity and would not contribute to the diagnostic accuracy).

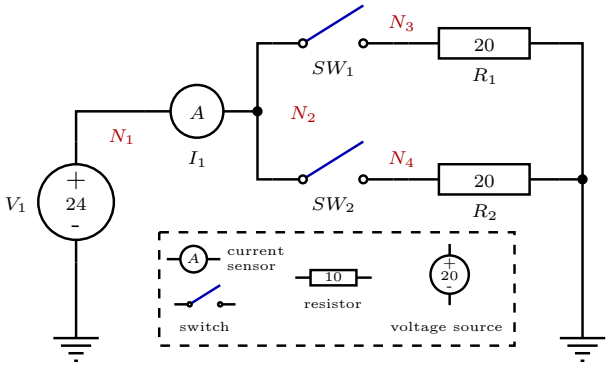

Figure 1. An electrical circuit used as a running example

The circuit in Fig. 1 consists of the voltage source V_1, a current sensor I_1, two normally closed switches SW_1 and SW_2 and the two resistors R_1, and R_2. Notice that if we know the state of the two switches SW_1 and SW_2, the rest of the circuit can be simulated with a circuit simulator like SPICE.

Models and Diagnoses

We start with the definition of a model. This is a very broad definition (our approach is agnostic to the modeling approach) - we say that a model is a function that computes some set of prediction variables. These prediction variables are observable from the viewpoint of diagnosis (i.e., they contain sensor data), hence we call them OBS as is traditional in MBD [11]. For our model to be useful for diagnosis and disambiguation, we distinguish two more special subsets of variables: the set of component variables COMPS and the set of command variables CMDS. Both COMPS and CMDS are from Finite Domain Integers (FDI) and their domains are typically small. This limitation is necessary for our AI-based search algorithms to work but limits our approach to relatively abrupt failures and limits our capabilities for graceful degradation and prognostics.

Definition 1 (System Model)**.** A system model M, $\langle$SD, CMDS, COMPS, OBS$\rangle$, is specified using a function SD from two sets of Finite Domain Integer (FDI) variables CMDS and COMPS, to a set of real-valued variables OBS.

In the above definition SD is a function that, when given *full* assignments over CMDS and COMPS, can compute an assignment for the variables in OBS. One can treat CMDS and COMPS as parameters or function selectors. Notice that the function should be specified in such a way so it would result in a valid OBS-assignment for *any* CMDS and COMPS instantiation. In MBD terminology SD is a strong-fault model [12].

The COMPS variables are called component, health, or assumable variables. The CMDS variables are command-variables or user-modifiable inputs (there may be also inputs that the user cannot modify but they have no special meaning for our algorithms). The COMPS-assignment we call a health assignment. The CMDS-assignment we call command-assignment.

We normally associate a *component* with one health variable and, if applicable, a command variable. A resistor, for example, is modeled only with a health variable, while a switch has a command variable that specified the position and a health variable that specifies if the switch is, for example, stuck.

Each health variable has a nominal mode, and the health assignment containing nominal modes only is called the nominal health assignment. In our running example, the nominal health assignment is $\omega^\star = \{V_1 = \text{nominal}, \text{I}_1 = \text{nominal}, \text{SW}_1 = \text{nominal}, \text{R}_1 = \text{nominal}, \cdots\}$. Note, that even though the health variables are FDI, we usually use symbols instead of numerals for readability. The representation in the implementation of the algorithms, of course, uses small numbers.

Similarly, we have command assignments. There is always a default value for each command variable. For our example in Fig. 1, the switches are normally closed, and the default command assignment is $\gamma = \{\text{SW}_1 = \text{closed}, \text{SW}_2 = \text{closed}\}$.

For the various health (and diagnosis) assignments that we use in this paper we use indexes of the Greek letter ω and for the command assignments we use γ. With all this we can define a hypothetical diagnostic scenario in we can "inject" a hypothetical health (fault) assignment $\omega^\star$.

Definition 2 (Diagnostic Scenario)**.** A diagnostic scenario SCN, $\langle M, \gamma, t^\star, \omega^\star, \alpha\rangle$, is specified using a system model M, an assignment $\omega^\star$ to the health variables in COMPS at time $t^\star$, and an assignment γ to the user-command variables in CMDS.

What is new in the above definition is the use of the observation assignment α. We assume that α is a real vector but keep the notation compatible with other AI papers where the observation is a Boolean assignment [13].

In what follows we define diagnosis.

Definition 3 (Diagnosis)**.** A diagnosis DIAG, $\langle M, \omega, t\rangle$ is defined as a probabilistic assignment ω to the variables in COMPS at time t.

A diagnosis is very similar to a health assignment, except that there is a probability for every literal. Let us say, that in our running example, the two of the resistors are open-circuited with the same probability of 0.25. The expression for ω looks like this: $\omega = \{\Pr(\text{R}_1 = \infty) = 0.25, \Pr(\text{R}_2 = \infty) = 0.25\}$.

In the definition of diagnostic scenario and diagnosis we also have the time of the fault injection $t^\star$ and the time of the diagnosis t. These are rarely used in our exposition as we construct most of the diagnostic scenarios in such a way as to have one fault injection and assume one diagnosis, which, if true (positive or negative), would have $t > t^\star$. We will need t and $t^\star$, though, when we define and compute the false-positive and false-negative scenario metrics.

Using the three definitions from above, we can essentially "run" diagnosis and disambiguation (we will shortly make the notion of disambiguation more precise). Clearly, the goal of a diagnostic system would be, given a diagnostic scenario, to compute a diagnosis ω that is close to $\omega^\star$. Of course, the diagnostic system cannot use the fault injection $\omega^\star$. The latter fault injection is only used while "training" and evaluating the whole diagnostic system. Finally, we need a function that will compute the "distance" between the fault injection $\omega^\star$ and the diagnosis ω. Such functions are well-studied in diagnosis, reliability, and others, and are called "metrics".

Definition 4 (Diagnostic Metric)**.** A diagnostic metric DM, $\langle F$, SCN, DIAG$\rangle$ is defined using a function F that maps $(t^\star, \omega^\star, t, \omega)$ into $[0, \infty)$.

The computation of metrics allows users to compare the performance of various diagnostic and simulation approaches, etc. The whole task of diagnosis and disambiguation can be cast as an optimization problem that aims at minimizing one or more metrics.

4. DIAGNOSIS

In what follows we show an algorithmic framework for computing diagnoses from models and scenarios. This is the first step in a two-step process in which (1) a diagnostics algorithm computes a diagnosis given a model and an observation and (2) if the diagnosis is ambiguous, i.e., one or more components are said to be in a certain fault state with probability different from one, then a disambiguation algorithm is used to decrease the diagnostic uncertainty. This section concerns the first step (diagnosis) while the next section describes the disambiguation process which involves the computation and application of new commands.

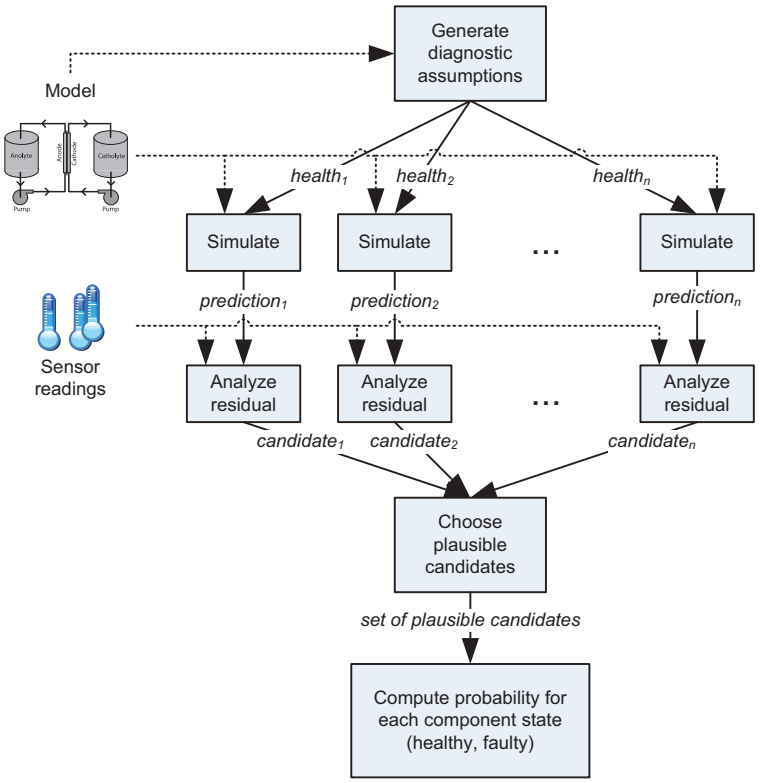

Figure 2. Overview of the LYDIA-NG diagnostic method

Overview

The basic idea of the LYDIA-NG diagnostic library (shown in Fig. 2) is to perform multiple simulations for various hypothesized health states of the plant. The output of these multiple simulations is then processed and combined into single diagnostic output.

The LYDIA-NG diagnostic library consists of the following building blocks:

Generator of Diagnostic Assumptions: A diagnostic assumption is a set of hypothetical assignments for the health or fault state of each component in the system. The "all nominal" diagnostic assumption assigns healthy status to each component. LYDIA-NG allows one nominal and one or more faulty states per component.
Simulation Engine: Given a diagnostic assumption, LYDIA-NG can construct a simulation model of the system. This simulation model consists of equations. By solving this system of equations LYDIA-NG computes values for one or more *observable* variables. The values of these observable variables is also referred to as a *prediction*.
Residual Analysis Engine: A prediction is compared to the sensor data by a residual analysis engine. This engine combines the individual discrepancies in each sensor data/predicted variable pair to produce a single real value that indicates how close is the prediction of the simulation engine to the sensor data obtained from the plant. A simulation that results in all predicted values coincide with the measured ones will result in the residual being zero. The data structure containing predictions, their corresponding sensor data and the computed residual is called a *diagnostic candidate* or simply *candidate*.
Candidate Selection Algorithm: Not all candidates generated by the residual analysis engine are used for computing the final system health. The candidate selection algorithm discards each candidate whose residual is larger than the residual of the "all nominal" candidate.
System State Estimation Algorithm: LYDIA-NG uses the set of candidates that is computed by the candidate selection algorithm to compute an estimate for the health of each component. This is done by the system state estimation algorithm. Finally, LYDIA-NG computes RCoF by choosing the components with highest probability of failure.

Algorithm

Algorithm 1 shows the top-level diagnostic process. The inputs to Alg. 1 are a model and a scenario, and the result is a diagnosis.

At the hear of Alg. 1 is the use of simulation. Algorithm 1 supports a large variety of simulation methods that may or may not use time as an independent variable. In the setup described in this paper we have used SPICE in combination with a constraint propagation solver. The latter we have used for sensor values, complex components such as mixed analog-digital electronics and other parts of the model where it is difficult or inappropriate to model with SPICE. The only requirement toward the simulation engine is to predict a number of variables whose types can be mapped to LYDIA-NG and to be relatively fast (the computational performance of LYDIA-NG will not be thoroughly discussed in this paper which emphasizes the application of LYDIA-NG to a space model).

The basic idea of Alg. 1 is to simulate for various health assignments and to compare the predictions with the observed sensor data (i.e., telemetry). There are several important

Algorithm 1 Diagnosis framework

```
1: function DIAGNOSE(SCN) returns a diagnosis
       inputs: SCN, diagnostic scenario
       local variables: h, FDI vector, health assignment
                        p, real vector, prediction
                        Ω, a set of diagnostic candidates
                        DIAG, diagnosis, result
2:     while h ← NEXTHEALTHASSIGNMENT() do
3:         p ← SIMULATE(M, γ, h)
4:         r ← COMPUTERESIDUAL(p, α)
5:         Ω ← Ω ∪ ⟨h, r⟩
6:     end while
7:     DIAG ← COMBINECANDIDATES(Ω)
8:     return DIAG
9: end function
```

aspects of this algorithms that ultimately affect the diagnostic accuracy as measured by various performance metrics (see Sec. 6).

The first algorithmic property that determines many of the diagnostic performances is the order in which health-assignments are generated. In Alg. 1 this is implemented in the NEXTHEALTHASSIGNMENT function. The latter subroutine also determines when to stop the search and should be properly parametrized depending on the model and the user requirements. In the standard LYDIA-NG diagnostic library we provide the following diagnostic search policies:

Breadth-First Search (BFS): This policy first generates the nominal health assignment, then single-faults, double-faults, etc.
Depth-First Search (DFS): This search policy starts with the nominal health assignment, then adds a single-fault, continues with a double fault including the first, and so on, until all components are failed. After the all-faulty assignment is generated, the algorithm backtracks one step and generates a sibling assignment and continues traversing down and backtracking in the same manner until no more backtracking is possible.
Backwards Greedy Stochastic Search (BGSS): In this mode, the search start from the all-faulty assignment. A random health variable is then flipped and the flip is retained iff the flip leads to a decrease in the residual. The order of health variables is arbitrary. As the whole search process is stochastic, it needs to be run multiple iterations in order to achieve the desired completeness. A formal description of this method for Boolean circuit models can be found in [13].

Each simulation produces what we call a *candidate*: a set of predicted values for a given health-assignment. The second important property of Alg. 1 is the comparison and ordering of the diagnostic candidates. This is done by mapping the predicted and observed variables into a single real-number, called *residual*. The residual computation is discussed in what follows.

Residual Generation

Residual generation functions in LYDIA-NG bear resemblance to loss functions in decision theory.

Definition 5 (Residual Function)**.** A residual function R maps a prediction vector $\mathbf{p}$ and an observation vector α into a real-number $[0; \infty)$.

We next show two straightforward residual generation functions.

Squared Residuals:

$$R_{\mathrm{sq}}(\mathrm{OBS}, \mathbf{p}, \alpha) = \sum_{\mathrm{v} \in \mathrm{OBS}} \mathrm{W(v)} \left[\mathbf{p}(\mathrm{v}) - \alpha(\mathrm{v})\right]^2 \qquad (1)$$

where $W(v)$ is a weight-value associated with sensor v, $\mathbf{p}(v)$ is the value of variable v in the prediction assignment $\mathbf{p}$ and $\alpha(v)$ is the value of the observable variable v.
Absolute Residuals:

$$R_{\mathrm{abs}}(\mathrm{OBS}, \mathbf{p}, \alpha) = \sum_{\mathrm{v} \in \mathrm{OBS}} \mathrm{W(v)} \left|\mathbf{p}(\mathrm{v}) - \alpha(\mathrm{v})\right| \qquad (2)$$

where $W(v)$, $\mathbf{p}(v)$ and $\alpha(v)$ are used in the same way as in Eq. 1.

A disadvantage of the squared residuals function R_{sq} is that it adds a lot weight to outliers. In decision theory, the absolute loss function that corresponds to the R_{abs} function is discontinuous. The latter, however, is not a problem for the algorithms described in this paper and we prefer R_{abs} over R_{sq}.

Computation of Component Failure Probabilities

Consider the circuit shown in Fig. 1 and a scenario $\alpha = \{I_1 = 1.19\}$. This scenario corresponds to one of the resistors being open-circuited or one of the switches being stuck-open. Table 1 shows applying Eq. 2 for the predictions simulated from the nominal and all single-fault health assignments. The rows of Table 1 are sorted in order of an increasing residual value. In this table (and below) we abbreviate a stuck switch as S and an open-circuit resistor mode as OC.

Table 1. Single-fault residuals for the circuit shown in Fig. 1 and an observation simulated from a single open-circuited resistor

V_1	I_1	SW_1	SW_2	R_1	R_2	faults	R_{abs}
–	–	S	–	–	–	1	0.0006
–	–	–	S	–	–	1	0.0006
–	–	–	–	OC	–	1	0.0006
–	–	–	–	–	OC	1	0.0006
–	–	–	–	–	–	0	1.1758
F	–	–	–	–	–	1	1.1888
–	F	–	–	–	–	1	1.1888
–	–	–	–	SC	–	1	79.3402
–	–	–	–	–	SC	1	79.3402

The COMBINECANDIDATES subroutine from Alg. 1 uses a table similar to the one shown in Table 1. It retains only the predictions with residuals smaller than the residual of the nominal prediction. The reason for that is that the nominal prediction is the only one that has a special meaning in LYDIA-NG and leads to a "landmark" residual, i.e., LYDIA-NG does not attempt to differentiate amongst the various fault-mode predictions. As a result, in our running example, only the first four rows of Table 1 are considered when calculating the final fault-probabilities.

The second step of COMBINECANDIDATES is to convert R_{abs} in the interval $[0; 1]$ where $R_{\mathrm{norm}} = 0$ for the nominal

Table 2. Normalized single-fault residuals from Table 1 that are smaller than the nominal residual

SW_1	SW_2	R_1	R_2	R_{norm}
S	–	–	–	1
–	S	–	–	1
–	–	OC	–	1
–	–	–	OC	1

prediction and $R_{\text{norm}} = 1$ for a fault prediction that gives $R_{\text{abs}} = 0$. Applying this on Table 1 gives us Table 2.

Finally, what remains to be done is to normalize the rightmost column of Table 2 so it sums up to one and marginalize the probability of failure in each column. For the small circuit we are analyzing this results in $\{\Pr(\text{SW}_1 = \text{S}) = 0.25, \Pr(\text{SW}_2 = \text{S}) = 0.25, \Pr(\text{R}_1 = \text{OC}) = 0.25, \Pr(\text{R}_2 = \text{OC}) = 0.25\}$. The fact that all probabilities are 0.25 means that Alg. 1 cannot determine unambiguously which component is the faulty one. In this case this is due to the fact that there is only one sensor, i.e., the unambiguity is due to sensor placement and circuit design.

One way to reduce this ambiguity is to change the position of SW_1 and/or SW_2. In the next section we devise an algorithmic framework that works for *any* circuit or model that can be diagnosed in the LYDIA-NG framework.

5. DISAMBIGUATION

An ambiguous scenario is when LYDIA-NG cannot be certain if a component is failing or not (we will shortly discuss a more precise notion of uncertainty). The reasons for that can be in the design of the system itself, due to model approximation, or due to the choice of the diagnostic algorithm. Consider, for example, the simple electrical circuit shown in Fig. 1. Due to the fact that there is a single current sensor in the design (I_1), an open-circuit resistor R_1 cannot be distinguished from an open-circuit R_2, SW_1, or SW_2. A set of modes that cannot be distinguished from each other is called an *ambiguity group*. Sometimes ambiguous results are due to the observation (consider the case in which there is no sensor data at all). In other cases, ambiguity is a result of the artifact design as shown in Fig. 1. In these cases no diagnostic algorithm can return a unique diagnosis. Finally, the *diagnosability* can be influenced by modeling approximation. Consider an alternative of the circuit shown in Fig. 1, in which R_1 is 22 Ω but it is modeled as a 20 Ω resistor.

Ambiguity groups appear only for certain plant configurations (positions of switches SW_1 and SW_2 in our example). Let us consider the case in which both SW_1 and SW_2 in Fig. 1 are closed and the only malfunctioning component is R_1, where R_1 is open-circuited. As a result of the fault, the current sensor I_1 shows -1.2 A[2] instead of the nominal -2.4 A. The most informed diagnosis in this case is that both R_1 and R_2 are equally-likely to be open-circuited. An equivalent statement is that given the single measurement of $\text{I}_1 = -1.2$ A, both R_1 and R_2 fail with probability of 0.5 (to keep the example short we do not allow SW_1 and SW_2 to fail, LYDIA, however, allows switches to fail).

[2]The current is negative so this document matches the LYDIA-NG implementation of the example and the passive sign convention [14].

A LYDIA-NG diagnosis contains a discrete probability for each component mode in the system. Each component typically specifies a single nominal mode and one or more fault modes. Consider, for example, R_1 that can be either in nominal mode ($R_1 = 20$), or in short-circuited mode ($R_1 = 0$), or in open-circuited mode ($R_1 = \infty$). We will use the following notation for the health of R_1:

$$\begin{array}{l} \Pr(\text{R}_1 = 20) = 0.5, \\ \Pr(\text{R}_1 = 0) = 0, \\ \Pr(\text{R}_1 = \infty) = 0.5 \end{array} \tag{3}$$

For brevity, we omit the zero-probability assignments:

$$\Pr(\text{R}_1 = 20) = 0.5, \Pr(\text{R}_1 = \infty) = 0.5. \tag{4}$$

In (4), the diagnostic engine does not really know if R_1 is healthy or open-circuited. A much more preferred situation from the diagnostic viewpoint would be:

$$\Pr(\text{R}_1 = \infty) = 1 \tag{5}$$

In (5) the diagnostic engine has determined a unique mode (R_1 is open-circuited), and there is no *uncertainty* in the diagnosis. We can derive a formula that gives a quantitative *metric* for the uncertainty in the health assignment of a component C:

$$U(C) = \sum_{x \in C^\star} -\Pr(\text{C} = \text{x}) \lg_{|\text{C}^\star|} \Pr(\text{C} = \text{x}), \tag{6}$$

where $\Pr(\text{C} = \text{x})$ denotes the probability of a component C being in a healthy/faulty state x and $C^\star$ is the set of all possible component states of C.

We can compute the uncertainty of a diagnostic assignment for the whole system as the average uncertainty of all of its components:

$$\bar{U} = \frac{1}{|\text{COMPS}|} \sum_{C \in \text{COMPS}} U(C), \tag{7}$$

where COMPS is the set of all components in the system.

If we treat each component C as a random variable (the sum of the probabilities of all component modes is one), then Eq. (6) represents the average component health *entropy* and Eq. (7) represents the average health entropy of the whole system. The values $U(C)$ and $\bar{U}$ are always in the interval $[0; 1]$ where 0 is the lowest uncertainty, and 1 is the highest, i.e., the probability for each component state is the same if and only if the entropy equals one.

Given an uncertain diagnosis, the LYDIA-NG disambiguation library computes new *plant-configuration assignment* (user-modifiable inputs) that optimally reduces the average uncertainty in the next diagnosis. The *next* (future) diagnosis, however, depends on a future observation which is unknown. To estimate this future observation, LYDIA-NG uses the existing (current) diagnosis.

Recall that a diagnosis is a probability assignment over the component states in the system. For a simulation, however, we need a deterministic assignment over the set of assumable (health) variables. For the simulation of a diagnosis, LYDIA-NG averages the observations obtained by failing all components that appear in the diagnosis with non-zero probability.

This is a weighted average and the weight for each simulation is the probability of failure as it appears in the diagnosis. By doing this, LYDIA-NG uses all information in a diagnosis for estimating a future observation.

Consider the circuit in Fig. 1 and a diagnosis

$$\Pr(\mathrm{R}_1 = \infty) = 0.5, \Pr(\mathrm{R}_2 = \infty) = 0.5 \quad (8)$$

The LYDIA-NG disambiguation algorithm first simulates the circuit with R_1 open-circuited, R_2 nominal, SW_1 closed, and SW_2 open. The predicted value for I_1 is 0 A. A simulation with R_1 nominal, R_2 open-circuited, SW_1 open, and SW_2 closed, results in the same predicted future observation of 0 A for I_1. As both simulated failures appear in (8) with the same probability the next predicted observation is $\mathrm{I}_1 = 0$ A. What remains for LYDIA-NG is to apply (6) to the diagnoses computed with the two observations just described. The first diagnosis is:

$$\Pr(\mathrm{R}_1 = \infty) = 1 \quad (9)$$

while the second diagnosis is:

$$\Pr(\mathrm{R}_2 = \infty) = 1 \quad (10)$$

Plugging (9) or (10) in (7) would maximize $\bar{U}$ which is intuitively correct as opening either SW_1 or SW_2 would result in current flowing through one resistor only, and hence disambiguate the resulting diagnosis. LYDIA-NG presents this kind of reasoning to the user who is advised to open SW_1 or SW_2 if she wants to obtain more diagnostic information than the one contained in (8).

A concluding remark is that by changing the positions of SW_1 and SW_2 we obtain a *virtual* (current) *sensor* that can be used for more precise determination of the health state of resistors R_1 and R_2.

LYDIA-NG can disambiguate in the case of *non-intermittent* failures only. Disambiguation of intermittent behavior is significantly more complicated and subject of future research. A subject of future development is also the automated computation of worst-case diagnosability of a model.

Algorithm 2 shows the LYDIA-NG disambiguation framework. It uses Alg. 1 as a diagnostic oracle.

The main loop of Alg. 2 (1) simulates the effects of a number of command assignments, (2) computes diagnosis and (3) takes the command assignment that results in a minimal diagnostic entropy.

When describing Alg. 2 we use *current* (diagnosis, observation, etc.) to denote the state before a new user command has been applied and *predicted* to denote observation, diagnoses, entropies, etc., after the new user command has been executed.

Algorithm 2 starts by computing the current diagnosis (line 2). The subroutine NEXTCOMMAND iterates over the space of all possible user commands. Notice that it takes as an argument the current state of all command variables and returns a new vector γ' that reflects the chosen command. If, for example, $\gamma = \{\mathrm{SW}_1 = \mathrm{closed}, \mathrm{SW}_2 = \mathrm{closed}\}$, executing the user command "open switch one", results in $\gamma' = \{\mathrm{SW}_1 = \mathrm{open}, \mathrm{SW}_2 = \mathrm{closed}\}$.

Depending on how NEXTCOMMAND is implemented, the following command generation strategies have practical significance:

Algorithm 2 Disambiguation framework

```
1:  function DISAMBIGUATE(SCN) returns a command
        inputs: SCN, diagnostic scenario
        local variables: DIAG, diagnosis
                         d, component health variable
                         h, FDI vector, health assignment
                         α, real vector, observation
                         γ', γ_min, command assignments
                         H, H_min, reals, entropy
2:      DIAG ← DIAGNOSE(SCN)
3:      while γ' ← NEXTCOMMAND(γ) do
4:          H ← 0
5:          for all {d ∈ DIAG : Pr(d) > 0} do
6:              h ← MAKEHEALTHASSIGNMENT(d)
7:              α ← SIMULATE(DM, γ, h)
8:              DIAG' ← DIAGNOSE(⟨M, γ', t, ω*, α⟩)
9:              H ← H + ENTROPY(DIAG')Pr(d)
10:         end for
11:         if H < H_min then
12:             H_min ← H
13:             γ_min ← γ'
14:         end if
15:     end while
16:     return γ_min
17: end function
```

Breadth-First Search (BFS): First, all possible changes to one command variable are considered, then all possible pairs, triples, etc. of command variables are generated. Clearly, this strategy quickly causes a combinational blow-up, hence for larger systems, it is feasible to consider only changes to single command variables.
Greedy Stochastic Search (GSS): This policy starts with a single change to a command variable. If this change leads to an entropy reduction (compared to the entropy of the current diagnosis), then this changed is preserved and a second command variable is changed. The process is repeated until changes to single command variables lead to a reduction in the predicted entropy. The order in which changes are applied is random, hence, the algorithm is stochastic. The disadvantage of this method is that it is incomplete, i.e., certain *combinations* of changes to command variables will not be considered given limited computational resources.

There are more command generation policies such as a fully stochastic generation of commands. This is hardly of practical interest as usually there is some cumulative cost related to combining individual changes to command variables into a multi-variable command and fully stochastic selection of changes would hardly result in minimal entropy.

One can also implement Backward Greedy Stochastic Search (BGSS). In this command generation strategy, the algorithm starts with changing all possible command variables in γ' and greedily flips the individual variables to their original values in γ. Again, a command variable flip should be accepted only if it reduces the predicted entropy. This strategy is suitable only for a class of devices that are configurable through multi-variable command changes.

In line 5 of Alg. 2 we iterate over each failing component d in the current diagnosis as computed by Alg. 1 in line 2 (each component that fails with non-zero probability). MAKEHEALTHASSIGNMENT (line 6) is an auxiliary function that returns a health assignment that has d failing with

its respective failure mode and all other component variables nominal. The current command assignment γ and the health assignment $\mathbf{h}$ are supplied to the simulator that is invoked in line 7. This is the same simulation engine that is used in Alg. 1. Finally, the ENTROPY function in line 9 is the implementation of Eq. 7.

The conditional in lines 11 - 14 select the user command γ' that minimizes the expected entropy H. This is collected int the $\gamma_{\min}$ variable which is returned in line 16. In the real world, the first k-lowest entropy γ' commands are presented to the user so she can choose the one that best matches the situation.

6. EXPERIMENTS

In this section we describe experiments with the LYDIA-NG framework in the scope of the GENIUS project. To make our study specific we have modeled the EPS of GOCE but the results are valid to a wider-class of satellite power-supply systems.

Exprimental Test-Bed

The experimental test-bed for LYDIA-NG is shown in Fig. 3. We have used the ESA-ESOC SIMSAT simulator framework to generate the failure scenarios for LYDIA-NG. GOCE uses the SCOS-2000 SCADA system and the LYDIA-NG diagnostic library interfaces it. The SIMSAT can be connected to SCOS-2000 via the Ground Models (a SIMSAT library that provides telemetry packet marshalling).

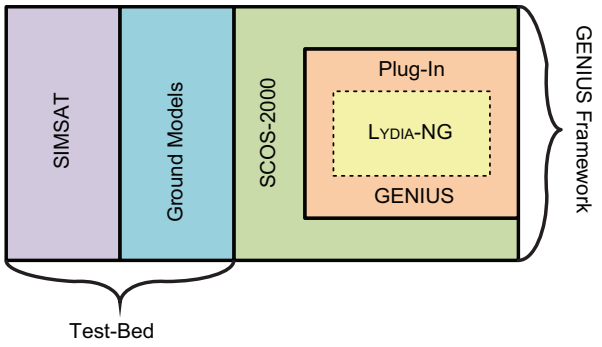

Figure 3. Overview of the GENIUS test-bed

GOCE EPS Model

From the viewpoint of modeling, the GOCE EPS has three major subsystems:

Solar Array Panels and Electronics: There are six body-mounted solar array panels. The panels are connected to the solar panel electronics and their voltage and current sensors are reported in the telemetry. Each solar array panel is connected to a Power Conditioning Unit (PCU). Each PCU contains, in addition to input and output voltage and current sensors, a Maximum Power Point Tracker (MPPT), and a solar array regulator. The PCU electronics is double and triple redundant. We have not considered scenarios leading to ambiguous diagnoses with faulty components in the PCU units.
Battery and Battery Control: The battery is connected to the main bus through a pair of switches that are not used in normal operation. There are redundant charging and discharging voltage and current sensors. The battery charge control electronics has the task of keeping the battery charged while extending its life.
Power Distribution Network and Thermal Control: There are two times eight heater groups (nominal and redundant), each group consisting of six heaters. Each heater can be individually controlled via a Transistor SWitch (TSW). The positions of the TSWs are reported in the telemetry. The heater groups are connected to the main bus through Latch Current Limiters (LCLs). There is a current sensor per heater group.

Figure 4 shows a high-level representation of the GOCE EPS. The LYDIA-NG model is $\approx$ 1000 lines and the number of variables is typically large for an MBD application (see Table 3).

Table 3. GOCE EPS model properties

Model Property	Value
observable variables	186
health variables	289
command variables	132
internal variables	790
total number of variables	1397

Diagnostic Scenarios

Table 4 shows a summary of the GENIUS diagnostic scenarios with which we have tested LYDIA-NG.

Table 4. GOCE EPS fault scenarios

Name	Class	Faults
TST1	nominal	–
TST2	single-fault	current sensor
TST3	single-fault, ambiguous	heater
TST4	continuation of TST3	heater
TST5	multiple-faults	current sensors
TST6	multiple-faults, related	current sensor and a switch
TST7	multiple-faults, ambiguous	solar arrays and heaters
TST8	multiple-faults, degradation	solar arrays
TST9	multiple-faults, accumulation	solar array and a voltage sensor
TST10	slow degradation	heater
TST11	intemittent	switch

We have arbitrarily chosen heater group A8 for experimentation—the results do not change if we choose another group or a combination of groups.

TST1: There is no fault injection during this scenario. This scenario is used to check the diagnostic algorithms for spurious diagnoses (false positive scenarios).
TST2: In this scenario a single PCU current sensor is failed. The fault-mode is out-of-bound value. Such a fault can be identified unambiguously by a diagnostic algorithm.
TST3: This scenario injects a single open-circuited heater in the first heater group. Due to the fact that there is only a single

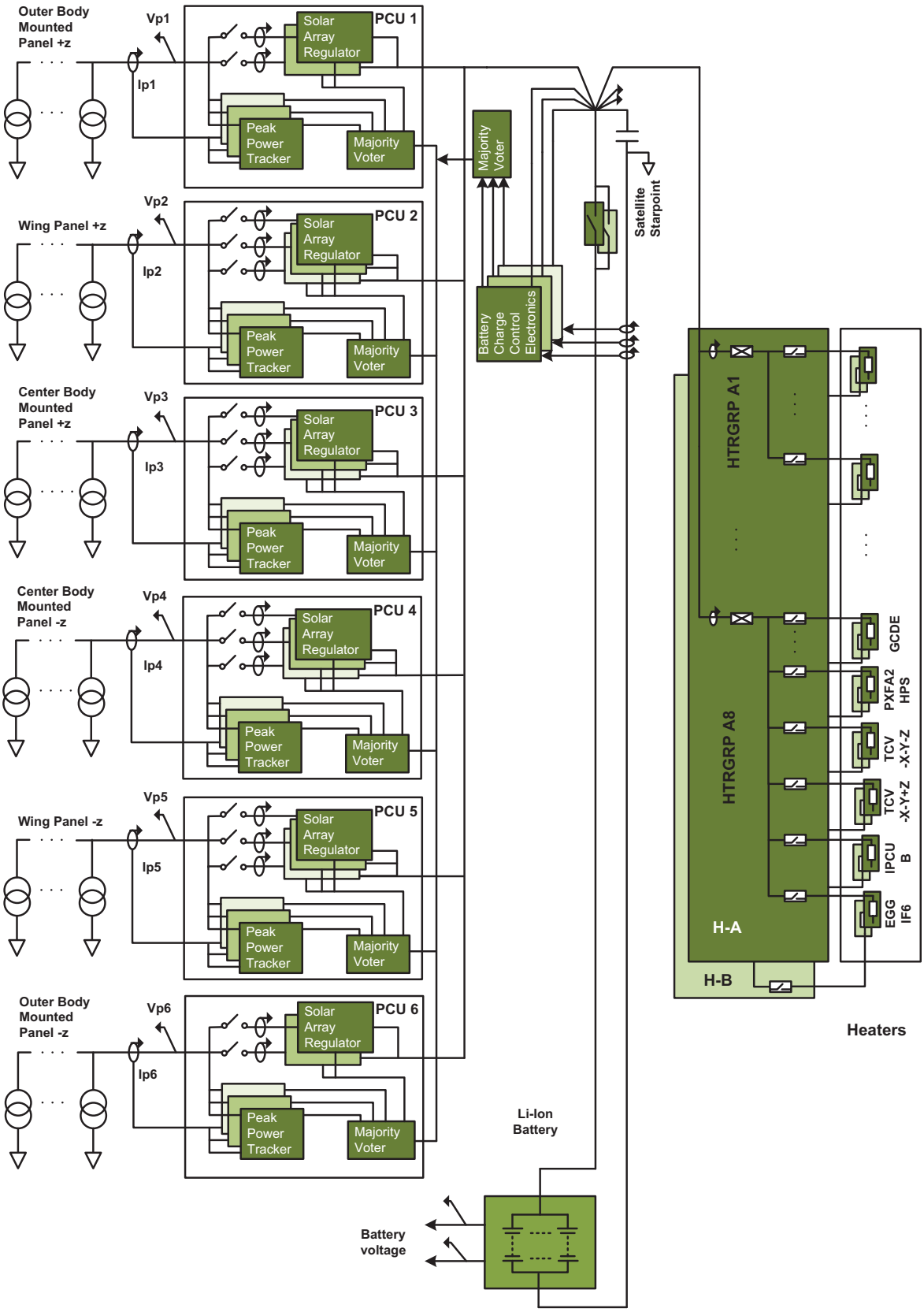

Figure 4. GOCE EPS

current-sensor per heater group and that there are heaters of the same type (resistance), this fault cannot be identified unambiguously by any diagnostic algorithm.
TST4: This is a continuation of TST3. After the first injection, the user is assumed to send a telecommand that switches-off one heater. This user-command now becomes part of the scenario. As a result a diagnostic algorithm should be able to produce a less-ambiguous diagnosis after the user-command.
TST5: In this scenario we inject simultaneously two current sensor faults of the same type as in TST2.
TST6: In this scenario we inject a double-fault. The two faults are a current sensor in the heater group and a switch. The two faults are topologically close, i.e., they are in the same heater group.
TST7: This is a quadruple-fault where the faults are relatively far from each other in the device topology. The scenario is highly-unlikely and is used for "stress-testing" the performance of the diagnostic and disambiguation algorithms. In this scenario we have two of the solar arrays suffer from the loss of multiple solar cells and two open-circuited heaters.
TST8: This is a double fault degradation, i.e., two of the solar-panels expose loss of power.
TST9: This scenario starts with degrading the capacity of a single solar-panel and then adds a failed voltage sensor. The individual faults accumulate and they are close topologically (i.e., one may have led to the other).
TST10: This scenario is not implemented at the time of the writing of this paper. It is included for planned future work on graceful-degradation, prognostics and other areas related to MBD. The idea is to have a heater that changes its resistance due to aging. At this moment the authors have not studied the exact physics of failure of such faults.
TST11: This is a scenario in which faults appear and disappear in consecutive measurements. The injected fault is a faulty LCL switch.

Metrics of Diagnostic Performance

The metrics for evaluating Diagnostic Algorithm (DA) performance depend on the particular use of the diagnostic system, the users involved, and their objectives.

Several institutions and organizations have proposed metrics that measure diagnostic performance [15], [16], [17], [18], [19], [20], [21]. Among those, the SAE's "Health and Usage Monitoring Metrics" [15] defines probability of detection and probability of false alarms as key indices for evaluating diagnostic algorithm performance.

In [19], the performance metrics are defined separately for detection, and isolation. For detection, the metrics include thresholds, accuracy, reliability, sensitivity to load, speed, or noise, and stability. The isolation metrics include the detection metrics, but also include measures for discrimination and repeatability.

In LYDIA-NG, we make a distinction between detection, isolation, and computational performance and highlight metrics for each category. In general several other classes of metrics are possible, including cost/utility metrics, effort metrics (in building systems for example) and also other categories such as fault identification and fault recovery metrics. The expectation is that as LYDIA-NG evolves a comprehensive list of desired metric classes and categories will be developed to aid framework users in choosing the performance criteria they want to measure.

The ten metrics we have defined are summarized in Table 5.

Table 5. Metrics summary

Metric	Name	Class
M_{fd}	fault detection time	detection
M_{fn}	false negative scenario	detection
M_{fp}	false positive scenario	detection
M_{da}	scenario detection accuracy	detection
M_{fi}	fault isolation time	isolation
M_{err}	classification errors	isolation
M_{ia}	isolation accuracy	isolation
M_{ent}	diagnostic uncertainty	isolation
M_{cpu}	CPU load	computational
M_{mem}	memory load	computational

All metrics are real numbers (M_{mem} is an exception as it measures a discrete value - the number of bytes that are used). Several metrics (M_{fn}, M_{fp}, M_{da}, M_{err}, M_{ia}, and M_{utl}) are defined in such a way as to produce a number in the interval $[0; 1]$. We show that isolation accuracy is dual to classification errors and, as M_{ia} does not contain any additional information to M_{err}, LYDIA-NG does not compute it.

Detection Metrics—The distinction between detection and isolation has practical importance. A DA may announce a fault detection before it knows the root cause of failure (for example, a detection announcement can be based solely on surpassing sensor threshold values). A detection signal cannot be retracted by a DA while it is legal to retract an isolation announcement when more faults are expected (at the of writing of this LYDIA-NG does not support retractions but such feature may be added in the future). The detection metrics include:

Fault Detection Time: The *fault detection time* (the reaction time for a diagnostic engine to detect an anomaly) is directly measured as:

$$M_{\mathrm{fd}} = t - t^{\star} \tag{11}$$

The fault detection time is reported in milliseconds and is computed only for non-nominal scenarios for which a DA computes a diagnosis at least once.
False Negative Scenario: The *false negative scenario* metric measures whether a fault is missed by a diagnostic algorithm and is defined as:

$$M_{\mathrm{fn}} = \begin{cases} 1, & \text{if t} = \infty \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

M_{fn} of a nominal scenario is defined to be zero.
False Positive Scenario: The *false positive scenario* metric penalizes DAs which announce spurious diagnoses and is defined as:

$$M_{\mathrm{fp}} = \begin{cases} 1, & \text{if t} < \text{t}^{\star} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

where $t^{\star} = \infty$ for nominal scenarios (i.e., scenarios during which no fault is injected).
Note that the above two metrics (M_{fn} and M_{fp}) are computed for each scenario and their computation is based on the times of injecting and announcing the fault. We also have false negative and false positive components in the context of individual diagnostic candidates (recall that a DA sends a

set of diagnostic candidates at isolation time) which we will discuss later in this document.
Scenario Detection Accuracy: The *scenario detection accuracy* metric is computed from M_{fn} and M_{fp}:

$$M_{\text{da}} = 1 - \max(M_{\text{fn}}, M_{\text{fp}}) \tag{14}$$

M_{da} is 1 if the scenario is true positive or true negative and 0 otherwise (equivalently, $M_{\text{da}} = 0$ if $M_{\text{fn}} = 1$ or $M_{\text{fp}} = 1$, and $M_{\text{da}} = 1$ otherwise). M_{da} splits all scenarios into "true" and "false". Incorrect scenarios are further classified into false positive (M_{fp}) and false negative (M_{fn}). Correct scenarios are true positive if there are injected faults and true negative otherwise (the latter separation into true positives and true negatives is rarely of practical importance).

Isolation Metrics—Consider a fault injection ω. We construct the *set* of failing components $\bar{\omega}$ by taking the identifiers of the failing components only. In other words, we discard the precise fault state of a failing component, and only retain the information that this component is in an off-nominal state.

We can do something similar from a diagnosis $\omega^\star$. Of course, in this case we will take only components that are *failing* with non-zero probability. We denote the set of failing components in a diagnosis ω as $\bar{\omega}$.

Notation-wise, we put a bar above a fault injection $\omega^\star$ or a diagnosis ω to take two sets of failing components - the fault $\bar{\omega}^\star$ and the candidate $\bar{\omega}$. We can now use those two to explain false positives and false negatives in the context of multiple-fault isolation accuracy metrics.

Both the candidate $\bar{\omega}$ and the injected fault $\bar{\omega}^\star$ are sets of components. The intersection of those two sets are the properly diagnosed components. The false positives are the components that have been considered faulty but are not actually faulty. The false negatives are the components that have been considered healthy but are actually faulty. Figure 5 shows how $\bar{\omega}$ and $\bar{\omega}^\star$ partition all components into four sets.

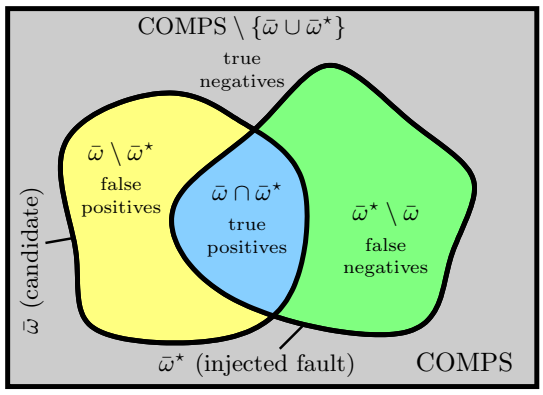

Figure 5. The diagnostic candidate $\bar{\omega}$ and the injected fault $\bar{\omega}^\star$ partition COMPS into four sets

False positives and false negatives in this context relate to individual candidates, i.e., misclassified components in a single diagnostic candidate. There are also scenario-based false negative and false positive metrics (defined earlier in this section), which summarize whole scenarios and are not to be confused with the false positives and false negatives in the context of isolation metrics.

For brevity, in the remaining of this document we use the notation in Table 6 for the Fig. 5 sets.

Table 6. Notation for sizes of some frequently used sets

Var.	Set	Description
f	$\lvert\text{COMPS}\rvert$	all components
n	$\lvert\bar{\omega}^\star \setminus \bar{\omega}\rvert$	false negatives
N	$\lvert\text{COMPS} \setminus \bar{\omega}\rvert$	the set of healthy components from the viewpoint of the DA
$\bar{n}$	$\lvert\bar{\omega} \setminus \bar{\omega}^\star\rvert$	false positives
$\bar{N}$	$\lvert\bar{\omega}\rvert$	the set of faulty components from the viewpoint of the DA

Based on the representation given in Figure 5, the meaning of false positives and false negatives can be interpreted differently depending on what the diagnosis results are supporting (abort decisions, ground support, fault-adaptive control, etc.). Researchers have proposed different methods to assess the meaning of isolation accuracy and its practical and economical implications.

[16] introduced metrics based on the receiving operating characteristic (ROC) analysis [18], which illustrates the trade-off space between the probability of false alarm and the probability of detection for different signal to noise ratio (SNR) levels. The method is used to test the relative accuracy of diagnostic systems based on different threshold settings. Later, they also proposed a combined metric [17] that accounts for consequential event costs including missed detection, false alarms, and misdiagnosis. Another widely used metric for isolation accuracy is the Kappa Coefficient [15]. It is based on the construction of a confusion matrix that summarizes diagnostic results produced by a reasoner over a number of test/use cases. In essence, the Kappa Coefficient measures the ability of an algorithm to discriminate among many fault candidates.

In this document, we take a simplistic approach and assume that false positives and false negatives have an equal cost for the diagnostic task and operations. The isolation metrics include:

We can now return to using the original $\omega^\star$ and ω, i.e., we don't need their set equivalents $\bar{\omega}^\star$ and $\bar{\omega}$.

Classification Error: Given a fault injection $\omega^\star$ and a diagnosis ω, the *classification error* metric is defined as:

$$M_{\text{err}} = \sum_{C \in \text{COMPS}} |I(C = x) - \Pr(\text{C} = \text{x})| \tag{15}$$

where

$$I(C = x) = \begin{cases} 1, & \text{if } (\text{C} = \text{x}) \in \omega \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

In Eq. (15), $|I(C = x) - \Pr(\text{C} = \text{x})|$ denotes the symmetric difference of the ω and $\omega^\star$ assignments and can be interpreted as the probabilistic equivalent of the number of misclassified components.

We can define a new metric, the isolation accuracy M_{ia}:

$$M_{\text{ia}} = \sum_{C \in \text{COMPS}} |1 - I(C = x) - \Pr(\text{C} \neq \text{x})| \qquad (17)$$

One can easily see that M_{ia} and M_{err} are duals, i.e.:

$$\frac{M_{\text{ia}}}{f} + \frac{M_{\text{err}}}{f} = 1 \qquad (18)$$

The isolation accuracy metric M_{ia} originates in the automotive industry [15]. The Aerospace Recommended Practice (ARP) computes the closely related probability of correct classification in the following way. For each component we compute the square confusion matrix. The probability of correct classification is the sum of the main diagonal divided by the total number of classifications (see the referenced ARP [15] for details and examples).

Diagnostic Uncertainty: Unlike M_{err} and M_{ia}, the diagnostic uncertainty metric M_{ent} is computed from the diagnosis ω only. As a result, the diagnostic uncertainty is not suitable for being used on its own, but should be used in combination with M_{ia} and M_{err}.

We start by defining the diagnostic uncertainty of a component $C \in \text{COMPS}$:

$$U(C) = \sum_{x \in C^\star} -\Pr(\text{C} = \text{x}) \lg_{|\text{C}^\star|} \Pr(\text{C} = \text{x}), \qquad (19)$$

where $\Pr(\text{C} = \text{x})$ denotes the probability of a component C being in a healthy/faulty state x and $C^\star$ is the set of all possible component states of C.

We can compute the uncertainty of a diagnostic assignment for the whole system as the average uncertainty of all of its components:

$$M_{\text{ent}} = \frac{1}{f} \sum_{C \in \text{COMPS}} U(C), \qquad (20)$$

where COMPS is the set of all components in the system.

Fault Isolation Time: Consider a fault injection $\omega^\star$ and a sequence of diagnoses $\omega_1, \omega_2, \ldots, \omega_n$. The fault isolation time M_{fi} is defined as:

$$M_{\text{fi}} = t' - t^\star \qquad (21)$$

where t' equals the first time t_i ($i = 1, 2, \ldots, n$) in which M_{ent} of ω_i is zero.

The fault isolation time is reported in milliseconds.

Results

Table 7 shows the main results of our experiments. As the full isolation requires a manual step (the selection of a user-command amongst the first lowest-entropy ones) and the simulation of the effect of the user-command we have shown only the first disambiguation step which leads to a lower diagnostic uncertainty.

All experiments were performed on a modern Pentium desktop (3 GHz), however, the implementation of the LYDIA-NG algorithms was single threaded. Further, we have used simulation databases, to improve the speed of the simulation process. The offline time for computing a simulation database is less than two hours. After that, the online step of looking-up the database and computing the residual takes less than 100 ms. These results show that LYDIA-NG can be used for real-time monitoring and diagnosis, processing telemetry data as it arrives, typically at a rate of 2 Hz.

The GOCE experiments showed no false-positive and false-negative scenarios, hence the diagnostic accuracy was one (see columns M_{fn}, M_{fp}, and M_{da} in Table 7). The classification errors and the isolation accuracy metrics are close to the optimal for the model, although the worst-case values of M_{err} and M_{ia} are not computed (i.e., the optimal values for the *worst-case* observation). The latter would be useful during the design stage and we plan to implement them in future work.

The relatively large memory consumption (511 MiB in all scenarios) is due to the use of non-compressed simulation databases. This is not a problem for ground (as opposed to on-board) as RAM for this kind of application is nowadays very cheap.

The fault isolation time M_{fi} is a worst-case estimation that includes the computation of disambiguation only. We assume that the user always chooses a command that does not lead to switching-off the faulty component (a user action that would masquerade the fault leads to a predicted uncertainty of zero and would be proposed by Alg. 2). Considering TST3, for example, the M_{fi} estimation is done assuming that the user switches-off all heater switches in the group before reaching the faulty one.

7. CONCLUSIONS

In this paper we have reported on the GENIUS project. The purpose of the project was to bring MBD one step closer to the real-world in assisting operators in satellite ground control.

In this project we have simulated telemetry that is close to real-world. We have defined a number of scenarios in which we have injected multiple, ambiguous, intermittent faults and we have defined a number of performance metrics with which we have tested diagnosis and disambiguation. Our approach is promising in that it delivers good scenario diagnostic accuracy, small number of classification errors. Further, the disambiguation approach we propose can reduce the isolation accuracy and assist operators in decision making and execution of (sequences of) telecommands.

There are several immediate continuations of this project. The first step would be to model additional satellite subsystem, and the best candidate would be the thermal subsystem. Testing with real (non-simulated) telemetry would also bring the technology further. Of course, obtaining telemetry of real faults should take into consideration security and privacy aspects.

REFERENCES

[1] S. Narasimhan and L. Brownston, "Hyde–a general framework for stochastic and hybrid model-based diagnosis," in *Proc. 18th International Workshop on Principles of Diagnosis (DX'07), Nashville, USA*, 2007, pp. 162–169.

[2] M. Daigle, I. Roychoudhury, G. Biswas, X. Koutsoukos, A. Patterson-Hine, and S. Poll, "A comprehensive diagnosis methodology for complex hybrid systems: A case study on spacecraft power distribution systems,"

Table 7. Diagnostic metrics from the GOCE EPS experiments

Name	M_{fd} [ms]	M_{fn}	M_{fp}	M_{da} [ms]	M_{fi} [s]	M_{err}	M_{ia}	M_{ent}	M_{cpu} [ms]	M_{mem} [MiB]
TST1	–	0	0	1	–	0	289	0	114	511
TST2	88	0	0	1	–	0	289	0	88	511
TST3	88	0	0	1	505	1.45	287.55	0.004	88	511
TST4	89	0	0	1	–	0	289	0	89	511
TST5	90	0	0	1	–	1	288	0.0035	90	511
TST6	90	0	0	1	101	1	288	0.0016	90	511
TST7	93	0	0	1	505	3.96	285.04	0.0047	93	511
TST8	94	0	0	1	–	1	288	0.0022	94	511
TST9	84	0	0	1	–	1	288	0	84	511
TST10	–	–	–	–	–	–	–	–	–	–
TST11	77	0	0	1	505	1.83	287.17	0.0025	77	511

Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on, vol. 40, no. 5, pp. 917–931, 2010.

[3] A. Zheng, I. Rish, and A. Beygelzimer, "Efficient test selection in active diagnosis via entropy approximation," *arXiv preprint arXiv:1207.1418*, 2012.

[4] S. Wu, Q. Yan, Y. Ren, and S. Guo, "Efficient probe prediction algorithm for fault diagnosis in computer networks," in *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*. IEEE, 2011, pp. 1–4.

[5] Y. Qiao, X. Qiu, L. Cheng, and L. Meng, "A methodology used to optimize probe selection for fault localization," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–5.

[6] G. Bellala, S. Bhavnani, and C. Scott, "Active diagnosis under persistent noise with unknown noise distribution: A rank-based approach," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[7] D. Tolpin and S. Shimony, "Rational value of information estimation for measurement selection," *arXiv preprint arXiv:1003.5305*, 2010.

[8] T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, L. Kuhn, J. de Kleer, A. van Gemund, and A. Feldman, "First international diagnosis competition - DXC'09," in *Proc. DX'09*, 2009, pp. 383–396.

[9] S. Poll, A. Patterson-Hine, J. Camisa, D. Garcia, D. Hall, C. Lee, O. Mengshoel, C. Neukom, D. Nishikawa, J. Ossenfort, A. Sweet, S. Yentus, I. Roychoudhury, M. Daigle, G. Biswas, and X. Koutsoukos, "Advanced diagnostics and prognostics testbed," in *Proc. DX'07*, 2007, pp. 178–185.

[10] A. Feldman, T. Kurtoglu, S. Narasimhan, S. Poll, D. Garcia, J. de Kleer, L. Kuhn, and A. van Gemund, "Empirical evaluation of diagnostic algorithm performance using a generic framework," *International Journal of Prognostics and Health Management*, pp. 1–28, 2010.

[11] A. Feldman and A. van Gemund, "A two-step hierarchical algorithm for model-based diagnosis," in *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI'06), Boston, Massachusetts, USA*, July 2006.

[12] J. de Kleer, A. Mackworth, and R. Reiter, "Characterizing diagnoses and systems," *Artificial Intelligence*, vol. 56, no. 2-3, pp. 197–222, 1992.

[13] A. Feldman, G. Provan, and A. van Gemund, "Approximate model-based diagnosis using greedy stochastic search," *Journal of Artificial Intelligence Research*, vol. 38, pp. 371–413, 2010.

[14] G. W. Roberts and A. S. Sedra, *SPICE*, ser. The Oxford Series in Electrical and Computer Engineering. Oxford University Press, 1996.

[15] S. A. P. S. H. M. Committee E-32, "Health and usage monitoring metrics, monitoring the monitor," Tech. Rep. ARP5783, February 2008.

[16] H. R. DePold, J. Siegel, and J. Hull, "Metrics for evaluating the accuracy of diagnostic fault detection systems," in *Proc. TURBO'04*, 2004.

[17] H. R. DePold, R. Rajamani, W. H. Morrison, and K. R. Pattipati, "A unified metric for fault detection and isolation in engines," in *Proc. TURBO'06*, 2006.

[18] C. E. Metz, "Basic principles of ROC analysis," *Nuclear Medicine*, vol. 8, no. 4, pp. 283–298, 1978.

[19] R. F. Orsagh, M. J. Roemer, C. J. Savage, and M. Lebold, "Development of performance and effectiveness metrics for gas turbine diagnostic technologies," in *Proc. AEROCONF'02*, vol. 6, 2002, pp. 2825–2834.

[20] M. Roemer, J. Dzakowic, R. F. Orsagh, C. S. Byington, and G. Vachtsevanos, "Validation and verification of prognostic health management technologies," in *Proc. AEROCONF'05*, 2005, pp. 3941–3947.

[21] M. Bartyś, R. Patton, M. Syfert, S. de las Heras, and J. Quevedo, "Introduction to the DAMADICS actuator FDI benchmark study," *Control Engineering Practice*, vol. 14, pp. 577–596, 2006.

BIOGRAPHY

Alexander Feldman is a postdoc at University College Cork and a visiting researcher at Ecole Polytechnique Fédérale de Lausanne (EPFL), Delft University of Technology and PARC (former Xerox PARC). He has obtained his Ph.D. (cum laude) in computer science/artificial intelligence and M.Sc. (cum laude) in parallel and distributed systems from the Delft University of Technology. He has published in leading conference proceedings and international journals covering topics in artificial intelligence, model-based diagnosis, and engineering. In cooperation with NASA Ames Research Center and PARC, Alexander Feldman has co-organized the International Diagnostic Competitions (DXC). Alexander Feldman's interest cover wide spectrum, including topics such as model-based diagnosis, automated problem solving, software and hardware design, design of diagnostic space applications, digital signal processing, and localization.

Helena Vicente de Castro is a Space Software Engineer at CGI (previously at Logica), the Netherlands. In 2004, she received a PhD in Flying and Handling Qualities from Cranfield University, UK, on the Blended-Wing-Body configuration, and in 1998 an MSc in aeronautical engineering from Universidade da Beira Interior, Portugal. After her PhD, Helena worked as researcher at the Technical University of Delft in the Netherlands, investigating pilot in-the-loop oscillations. Later, at the European Space Agency, Helena developed fault detection and isolation filters based on robust control theory for satellites and re-entry vehicles. She presently works on testing the security chains of the European Global Navigation Satellite System Galileo. Helena's research interests are in the area of fault detection and isolation, automatic control systems, modelling and simulation, and more recently RAMS and security.

Arjan van Gemund received a BSc in Physics, an MSc degree (cum laude) in Computer Science, and a PhD (cum laude), all from Delft University of Technology. He has held positions at the R & D organization of the Dutch multinational company DSM as an Embedded Systems Engineer, and at the Dutch TNO Research Organization as a High-Performance Computing Research Scientist. From 1992 he was at the Electrical Engineering, Mathematics, and Computer Science Faculty of Delft University of Technology, the last 6 years serving as Full Professor in the area of Fault Diagnosis of Embedded Hardware and Software Systems. He has (co)authored over 200 scientific publications, and is (co)recipient of 9 best paper awards.

Gregory Provan is a Professor at the Computer Science Department at University College Cork (UCC), in Cork, Ireland. He received his BSE from Princeton University, USA, his MSc from Stanford University, USA, and his DPhil from University of Oxford, UK. He is currently Director of the Complex Systems Lab (`http://www.cs.ucc.ie/ccsl/`) at UCC. Prior to joining UCC he was on faculty at the University of Pennsylvania, USA, and spent over 10 years in industrial research. His interests are in complex systems (design, analysis and control), diagnostics, algorithm design, and bioinformatics. His current research focuses on modelling and analysis of sustainable energy systems, model-based diagnostics of a variety of complex systems, and design of methods for efficiently embedding code in such systems. Prof. Provan is the co-author of over 100 refereed articles, and has been Principal Investigator in a variety of grants and research contracts.