Diagnostic Reasoning using Prognostic Information for Unmanned Aerial Systems

Johann Schumann[1], Indranil Roychoudhury[2], and Chetan Kulkarni[3]

[1,2,3] *Stinger Ghaffarian Technologies, Inc., NASA Ames Research Center, Moffett Field, CA, 94040, USA*

{*johann.m.schumann, indranil.roychoudhury, chetan.s.kulkarni*}*@nasa.gov*

ABSTRACT

With increasing popularity of unmanned aircraft, continuous monitoring of their systems, software, and health status is becoming more and more important to ensure safe, correct, and efficient operation and fulfillment of missions. The paper presents integration of prognosis models and prognostic information with the R2U2 (REALIZABLE, RESPONSIVE, and UNOBTRUSIVE Unit) monitoring and diagnosis framework. This integration makes available statistically reliable health information predictions of the future at a much earlier time to enable autonomous decision making. The prognostic information can be used in the R2U2 model to improve diagnostic accuracy and enable decisions to be made at the present time to deal with events in the future. This will be an advancement over the current state of the art, where temporal logic observers can only do such valuation at the end of the time interval. Usefulness and effectiveness of this integrated diagnostics and prognostics framework was demonstrated using simulation experiments with the NASA Dragon Eye electric unmanned aircraft.

1. INTRODUCTION

For safe and efficient operation and successful fulfillment of missions, it is imperative for modern autonomous systems, such as unmanned aerial systems (UAS), to detect, in flight, if all their components are working in a nominal mode, or if there are any faults that might hamper their performance, endanger their missions, or even lead to crashes. Hence, continuous monitoring of the UAS systems, their software, and health status is becoming—even for small UAS—more and more important.

Traditional Fault Detection and Diagnosis (FDD) systems use the current system status as provided by on-board sensors to detect faults and perform root cause analysis. Many different FDD systems and approaches exist and are being used for commercial and military aircraft, automobiles, or complex (chemical) plants, e.g., (Frank, 1996). Many diagnostic reasoners that are used for on-board diagnostics are oblivious of any temporal relationships and strictly focus on the current state of the system, or need signal-preprocessing libraries (e.g., TEAMS RT (Mathur, Deb, & Pattipati, 1998)). Other systems like FACT (Karsai et al., 2006) allow the modeler to specify properties of temporal propagation that can be used for diagnosis. For example, if component A fails some time after component B and there is structural relationship, then the reasoner can deduce from the order of the events what actually happened. Again, all information is deduced from the current and past states of the UAS. The R2U2 (REALIZABLE, RESPONSIVE, and UNOBTRUSIVE Unit) Framework (Schumann, Rozier, et al., 2013; Schumann et al., 2015; Reinbacher, Rozier, & Schumann, 2014) is a diagnosis framework that combines observers for Metric Temporal Logic with Bayesian networks for probabilistic reasoning and root cause analysis. R2U2 is implemented in FPGA hardware (Geist, Rozier, & Schumann, 2014).

The performance and safety of an electrically powered UAS strongly depends on its battery. In most cases, high-powered rechargeable cells (e.g., Li-poly type cells) are used to power engines, electronics, and payload. When the battery gets depleted, its voltage starts decreasing and available power to the engine might be reduced. This can lead to lower speeds and climb rates. If the voltage drops below a certain threshold, the flight computer will stop and the UAS will crash. Prognostic information that gives a prediction of how much time remains before the battery voltage will drop below a certain threshold can be very valuable in such scenarios, ensuring even safer operation by enabling possible mitigation actions.

In this paper, we extend R2U2 to incorporate the use of prognostic models and reasoning information to take future predictions of safety and performance into account. This enables decision making at the present point in time. With this

extension, we are able to substantially improve model expressiveness with respect to future events. For example, a safety rule might say that "*the UAS state is only healthy if an upcoming climb is performed with enough battery reserves when the climb starts*". Assuming that the flight plan entails a climb in 10 minutes, a temporal encoding of that flight rule could be: "*within the next 10 minutes always have a good battery*." However, the validity of this formula can only be established after 10 minutes, unless the battery becomes depleted before that. That time, however, in general is too late to re-plan the mission. The synchronous R2U2 observers provide additional instantaneous information if it is still possible ("maybe") to observe the flight rule, but again, its ultimate validity can only be decided after 10 minutes. Our proposed extension allows for such a flight rule to be evaluated at the current time indicating if the battery will be good in 10 minutes.

With our novel incorporation of prognostics reasoning into the R2U2 framework, we will have information about the expected future battery performance at the present time, and the system can decide immediately if the flight rule will be violated in the future based upon these predictions. Obviously, assumptions about the battery performance and the estimated load profile are necessary. Using simulation experiments with the NASA Dragon Eye unmanned aircraft we will demonstrate how the use of an advanced prognostics model for the main UAS battery can augment and improve the UAS health models.

The rest of this paper is structured as follows: Section 2 presents an overview of the R2U2 framework with temporal observers and Bayesian reasoning and its realization in FPGA hardware. This section also gives an introduction into our model-based prognostics architecture. Section 3 focuses on the integration of prognostic reasoning into R2U2. In Section 4 we present results of simulation experiments with a NASA UAS and a prognostics model for battery performance. Section 5 discusses future work and concludes this paper.

2. BACKGROUND

In this section we present an overview over the R2U2 framework with its FPGA implementation and provide an introduction into prognostics.

2.1. The R2U2 Framework

Developed to continuously monitor system and safety properties of an UAS in flight, our real-time R2U2 (REALIZABLE, RESPONSIVE, and UNOBTRUSIVE Unit) framework has been implemented on FPGA (Field Programmable Gate Array) hardware (Reinbacher et al., 2014; Geist et al., 2014). Health models within this framework (Schumann, Rozier, et al., 2013; Schumann et al., 2015) are defined using Metric Temporal Logic (MTL) and Mission-time Linear Temporal Logic (LTL) (Reinbacher et al., 2014) for expressing temporal properties as well as Bayesian Networks (BN) for probabilistic and diagnostic reasoning.

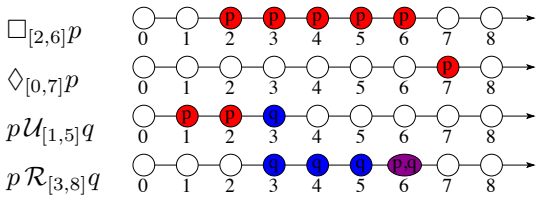


Figure 1. Illustration of MTL operators (from (Schumann, Rozier, et al., 2013)). Time steps and interval boundaries refer to ticks of the system clock.

R2U2 models are constructed in a modular way where outputs of sensor discretization and reasoning components can be connected to other monitoring and reasoning components (Schumann, Rozier, et al., 2013). An R2U2 model is usually specified using a graphical block representation and consists of data processing blocks, temporal logic observer blocks, and Bayesian diagnostic reasoning blocks. Additional block types provide utility functions.

2.1.1. Temporal Logic Monitors

LTL and MTL formulas consist of propositional variables, the Boolean operators $\wedge$, $\vee$, $\neg$, or $\rightarrow$, and MTL operators. For formulas p, q, we have $\Box p$ (ALWAYS p), $\Diamond p$ (EVENTUALLY p), $\mathcal{X}p$ (NEXTTIME p), $p\mathcal{U}q$ (p UNTIL q), and $p\,\mathcal{R}q$ (p RELEASES q). For MTL, each of the temporal operators is accompanied by an upper and lower time bound that express the time period during which the operator must hold. Figure 1 illustrates the MTL operators: $\Box_{[2,6]}p$ means that p must be true at all times between time step 2 and 6 along the time line (red dots). $\Diamond_{[0,7]}p$ means that p must be true at least once in the interval $[0, 7]$. In Figure 1, p is true at $t = 7$ making that formula true. $p\mathcal{U}_{[1,5]}q$ signifies that either q is true at the beginning of the interval, or else p is true at the beginning of the interval and will remain true until a future time (here: $t = 3$) within the interval, when q must be true (blue dot). Finally, $p\,\mathcal{R}_{[3,8]}q$ means that either $p \wedge q$ is true at the beginning of the interval or q is true then until a future time within the interval when both are true (purple dot at $t = 6$ in Figure 1). This operator is works like a push-button: pressing p triggers event $\neg q$ that "releases q" in the future. A detailed definition and semantics can be found in (Reinbacher et al., 2014).

2.1.2. Bayesian Networks for Health Models

In many situations, temporal logic monitoring might come up with several violations of properties. In order to be able to disambiguate the root causes, the R2U2 framework uses

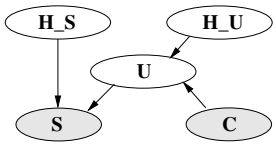


Figure 2. BN for Health management. Observable nodes are shaded.

static Bayesian Networks (BN) for diagnostic reasoning. BNs are directed acyclic graphs, where each node represents a statistical variable (Figure 2). BNs are well-established in the area of diagnostic and health management (e.g., (Pearl, 1985; Mengshoel et al., 2010)). Conditional dependencies between statistical variables are represented by directed edges. Local conditional probabilities are stored in the Conditional Probability Table (CPT) of each node. For example, the CPT of node S in Figure 2 defines $P(S|U, H\_S)$.

For our health models, we are using BNs of a general structure as shown in Figure 2. Discrete sensor signals or outputs of the synchronous temporal observers (`true`, `false`, `maybe`) are clamped to the S (sensor) and C (command) nodes. Since a sensor can fail, it has an unobservable health node attached.[1] As priors, these health nodes can contain information on how reliable the component is, e.g., by using a Mean Time To Failure (MTTF) metric. Unobservable nodes, U, may describe the behavior of the system or component as it is defined and influenced by the sensor or software information. For details of modeling see (Schumann, Mbaya, et al., 2013).

During flight, the posterior probabilities of the BN's health nodes, given the sensor and command values $\mathbf{e}$ as evidence are calculated at each time step. The probability $Pr(H\_S = good|\mathbf{e})$ gives an indication of the status of the sensor or component. For on-board real-time reasoning, we use a representation of BNs that is based upon arithmetic circuits (AC), as these data structures and algorithms provide predictable real-time performance (Chavira & Darwiche, 2005; Mengshoel et al., 2010).

2.2. Hardware Implementation

R2U2 is implemented as a separate hardware component. Figure 3A shows the high-level architecture of R2U2 integrated in a UAS. Controlled by an on-board flight computer and the flight software (FSW), the UAS receives measurements from various sensors (e.g., inertial sensors, GPS, or barometric altitude) and commands from the ground control station (GCS) and calculates the necessary adjustments of the actuators: elevator, rudder, ailerons, and throttle. Our R2U2 monitor obtains information from sensors and the FSW. Instrumentation of the FSW with small footprint (Geist et al., 2014; Schumann, Rozier, et al., 2013) guarantees minimal obtrusiveness of our framework. All monitoring data are transferred via a read-only interface into our R2U2 implementation that is hosted on an Adapteva Parallella board[2] mounted on the UAS.

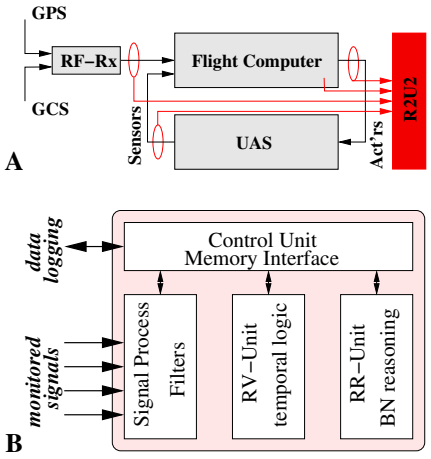


Figure 3. A: High level system architecture with R2U2. B: FPGA architecture.

Figure 3B shows the major components of the R2U2 monitoring unit as implemented in the FPGA: the control subsystem, the signal processing and filtering system (SP), the runtime verification (RV) unit, and the runtime reasoning (RR) unit. Continuous signals as obtained via the read-only interface are filtered and discretized in the SP unit to obtain streams of propositional variables. The RV and RR units comprise the proper health management hardware: RV monitors MTL properties using pairwise observers defined and proved correct in (Reinbacher et al., 2014). After the temporal logic formulas have been evaluated, the results are transferred to the runtime reasoning (RR) subsystem, where the compiled Bayesian network is evaluated to yield the posterior marginals of the health nodes (Geist et al., 2014).

2.3. Prognostics

In this section we discuss our developed electro-chemical model and battery prognosis framework following the general estimation-prediction framework of model-based prognostics (Luo, Pattipati, Qiao, & Chigusa, 2008; Orchard & Vachtsevanos, 2009; Daigle & Goebel, 2013). Details of the specific algorithms are described in (Daigle & Kulkarni, 2013). Similar approaches have been used for prognosis of pneumatic valves (Daigle, Kulkarni, & Gorospe, 2014; Kulkarni, Daigle, Gorospe, & Goebel, 2014) and for Current/Pressure (I/P) Transducers (IPT) (Teubert & Daigle, 2013, 2014) Here, we only summarize the formulation of the prognostics problem, followed by a brief description of the estimation and prediction approach.

[1] In contrast, am command input cannot fail and there for does not have an associated health node.

[2] http://www.adapteva.com/parallella-board/

2.3.1. Problem Formulation

We assume the system model may be generally defined as

$$\mathbf{x}(k+1) = \mathbf{f}(k, \mathbf{x}(k), \boldsymbol{\theta}(k), \mathbf{u}(k), \mathbf{v}(k)),$$
$$\mathbf{y}(k) = \mathbf{h}(k, \mathbf{x}(k), \boldsymbol{\theta}(k), \mathbf{u}(k), \mathbf{n}(k)),$$

where k is the discrete time variable, $\mathbf{x}(k) \in \mathbb{R}^{n_x}$ is the state vector, $\boldsymbol{\theta}(k) \in \mathbb{R}^{n_\theta}$ is the unknown parameter vector, $\mathbf{u}(k) \in \mathbb{R}^{n_u}$ is the input vector, $\mathbf{v}(k) \in \mathbb{R}^{n_v}$ is the process noise vector, $\mathbf{f}$ is the state equation, $\mathbf{y}(k) \in \mathbb{R}^{n_y}$ is the output vector, $\mathbf{n}(k) \in \mathbb{R}^{n_n}$ is the measurement noise vector, and $\mathbf{h}$ is the output equation.[3]

In prognostics, we predict the occurrence of an event E that is defined with respect to the states, parameters, and inputs of the system. We define the event as the earliest instant that some event threshold $T_E : \mathbb{R}^{n_x} \times \mathbb{R}^{n_\theta} \times \mathbb{R}^{n_u} \to \mathbb{B}$, where $\mathbb{B} \triangleq \{0, 1\}$ changes from the value 0 to 1. That is, the time of the event k_E at some time of prediction k_P is defined as

$$k_E(k_P) \triangleq \inf\{k \in \mathbb{N} \colon k \geq k_P \wedge T_E(\mathbf{x}(k), \boldsymbol{\theta}(k), \mathbf{u}(k)) = 1\}.$$

The time remaining until that event, Δk_E, is defined as

$$\Delta k_E(k_P) \triangleq k_E(k_P) - k_P.$$

For system health management, T_E is defined via a set of performance constraints that define what the acceptable states of the system are, based on $\mathbf{x}(k)$, $\boldsymbol{\theta}(k)$, and $\mathbf{u}(k)$ (Daigle & Goebel, 2013). For batteries, we are interested in end of discharge (EOD) time, i.e., the time at which the battery voltage will deplete below the voltage threshold V_{EOD}.

Models of the system components are constructed in this paradigm that capture both nominal behavior, as well as faulty behavior and damage progression. Using these models, observations can be mapped back to the health state of the system as represented in $\mathbf{x}$ and $\boldsymbol{\theta}$. An estimation algorithm, such as the Kalman filter (KF), unscented Kalman filter (UKF), or particle filter (PF), is used to solve this problem (Daigle, Saha, & Goebel, 2012). In this paper we use the UKF. This state-parameter estimate, along with a prediction of the future usage of the component, is used as input to a prediction algorithm that computes the time to EOD. This time is known as end of life (EOL), the difference between EOL and current time is called the remaining useful life (RUL) (Daigle & Goebel, 2013; Daigle, Saxena, & Goebel, 2012).

2.3.2. Prognostics Architecture

In our model-based prognostics architecture (Daigle & Goebel, 2013), there are two sequential problems, (*i*) the *estimation* problem, which requires determining a joint state-parameter estimate $p(\mathbf{x}(k), \boldsymbol{\theta}(k) | \mathbf{y}(k_0{:}k))$ based on the history of observations up to time k, $\mathbf{y}(k_0{:}k)$, and (*ii*) the *prediction* problem, which determines at k_P, using $p(\mathbf{x}(k), \boldsymbol{\theta}(k) | \mathbf{y}(k_0{:}k))$, a probability distribution $p(k_E(k_P) | \mathbf{y}(k_0{:}k_P))$. The distribution for Δk_E can be trivially computed from $p(k_E(k_P) | \mathbf{y}(k_0{:}k_P))$ by subtracting k_P.

The prognostics architecture is shown in Figure 4. In discrete time k, the system is provided with inputs $\mathbf{u}_k$ and provides measured outputs $\mathbf{y}_k$. The estimation module uses this information, along with the system model, to compute an estimate $p(\mathbf{x}(k), \boldsymbol{\theta}(k) | \mathbf{y}(k_0{:}k))$. The prediction module uses the joint state-parameter distribution and the system model, along with hypothesized future inputs, to compute the probability distribution $p(k_E(k_P) | \mathbf{y}(k_0{:}k_P))$ at given prediction times k_P.

2.3.3. Estimation

For batteries a detailed physics model of component behavior using nominal data from the testbed has been developed, which is discussed in (Daigle & Kulkarni, 2013). We use an unscented Kalman filter (UKF) to obtain the state estimate from the sensor measurements, as described in (Daigle & Kulkarni, 2013).

Battery Modeling: In order to predict end-of-discharge (EOD) as defined by a voltage cutoff, the battery model must compute the voltage as a function of time given the current drawn from the battery. There are several electrochemical processes that contribute to the cell's potential that make this a difficult problem. For the purposes of on-line prognostics, we focus here on a lumped-parameter ordinary differential equations form that still considers the main electrochemical processes. We focus on Li-ion 18650 batteries with an average nominal voltage of 4.2V and nominal capacity of 2200mAh.

The voltages of a battery are summarized in Figure 5 (adapted from (Rahn & Wang, 2013)). The overall battery voltage $V(t)$ is the difference between the potential at the positive current collector, $\phi_s(0, t)$, and the negative current collector, $\phi_s(L, t)$, minus resistance losses at the current collectors (not shown in the diagram). As shown in the figure, the potentials vary with the distance $d \in [0, L]$, because the loss varies with distance from the current collectors. The details of the battery model are discussed in (Daigle & Kulkarni, 2013).

State of Charge: State of Charge (SOC) of a battery is conventionally defined to be 1 when the battery is fully charged and 0 when the battery is fully discharged. In this model, it is analogous to the mole fraction x_n, but scaled from 0 to 1. There is a difference here between nominal SOC and *apparent* SOC. Nominal SOC would be computed based on the combination of the bulk and surface layer CVs in the negative electrode, whereas apparent SOC would be computed based only on the surface layer. That is, a battery can be discharged at a given rate, and reach the voltage cutoff, i.e.,

[3] Bold typeface denotes vectors, and n_a denotes the length of a vector $\mathbf{a}$.

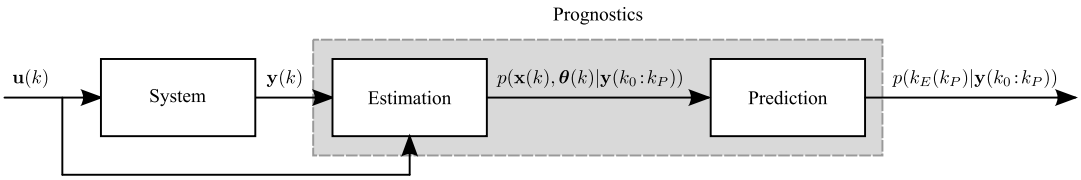


Figure 4. Prognostics architecture (see (Daigle & Goebel, 2013))

apparent SOC is then 0. But, once the concentration gradient settles out, the surface layer will be partially replenished and the battery can be discharged further, i.e, apparent SOC increases whereas nominal SOC remains the same.

Nominal (n) and apparent (a) SOC can then be defined using

$$SOC_n = \frac{q_n}{0.6q^{\max}} \tag{1}$$

$$SOC_a = \frac{q_{s,n}}{0.6q^{\max_{s,n}}}, \tag{2}$$

where $q^{\max_{s,n}} = q^{\max}\frac{v_{s,n}}{v_n}$. The factor $1/0.6$ comes from the fact that the mole fraction at the positive electrode cannot go below 0.4 (Daigle & Kulkarni, 2013), therefore SOC of 1 corresponds to the point where $q_n = 0.6q^{\max_{s,n}}$.

Battery Voltage: Now that each of the voltage drops in Figure 5 have been defined, battery voltage can be expressed as follows.

$$V = V_{U,p} - V_{U,n} - V_o - V_{\eta,p} - V_{\eta,n}. \tag{3}$$

Voltages in the battery are not observed to change instantaneously, i.e., the voltage change occurs smoothly. When discharge completes, for example, the voltage rises slowly as the surface layers move to the concentrations of the bulk volumes, as caused by diffusion. In addition to this, there are transients associated with V_o and the $V_{\eta,i}$ terms. To take this into account in a simple way, we compute voltage using

$$V = V_{U,p} - V_{U,n} - V'_o - V'_{\eta,p} - V'_{\eta,n}, \tag{4}$$

where

$$\dot{V}'_o = (V_o - V'_o)/\tau_o \tag{5}$$

$$\dot{V}'_{\eta,p} = (V_{\eta,p} - V'_{\eta,p})/\tau_{\eta,p} \tag{6}$$

$$\dot{V}'_{\eta,n} = (V_{\eta,n} - V'_{\eta,n})/\tau_{\eta,n}, \tag{7}$$

where the τ parameters are empirical time constants.

The model contains as states $\mathbf{x}$, $q_{s,p}$, $q_{b,p}$, $q_{b,n}$, $q_{s,n}$, V'_o, $V'_{\eta,p}$, and $V'_{\eta,n}$. The single model output is V.

2.3.4. Prediction

For the batteries, we simulate for various state of charge (SOC) values and load values the corresponding remaining time until discharge, and compute a lookup table. Given the SOC, as computed by the UKF, and expected future load, we can then quickly compute the corresponding time of EOD.

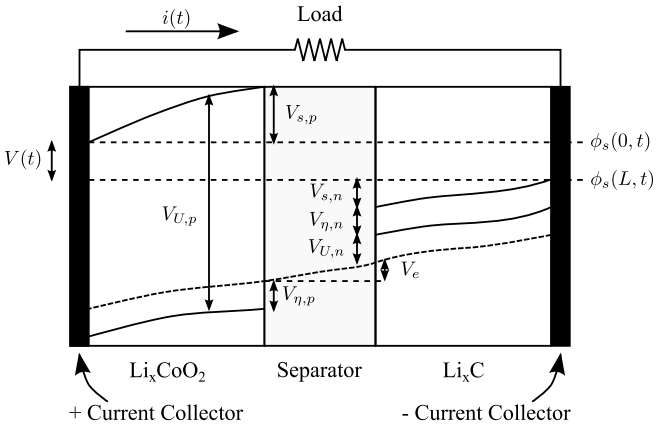


Figure 5. Battery voltages.

2.3.5. Application to Prognostics

With an accurate model and known future inputs to a system, prognostics should in turn be accurate. We use the architecture described in Section 2.3. As an estimation algorithm, we use the unscented Kalman filter (UKF) with the battery model; see (Julier & Uhlmann, 1997, 2004) for details on the filter and (Daigle, Saha, & Goebel, 2012; Daigle, Saxena, & Goebel, 2012) for its application to prognostics. The UKF operates on a set of deterministically selected samples, called *sigma points*, that are used to represent the joint state-parameter distribution $p(\mathbf{x}(k), \boldsymbol{\theta}(k)|\mathbf{y}(k_0{:}k))$.

For the prediction algorithm, we perform a simple simulation as described in (Daigle & Goebel, 2013). Each sigma point is simulated forward using the model until EOD is reached; from the corresponding EODs for each sigma point we can construct the EOD distribution. In this work, we assume that the future inputs (i_{app}) are known, so the only uncertainty present in the prediction is that related to the model. A defined cutoff voltage is assumed to define EOD.

3. APPROACH

We integrate prognostics capabilities into our R2U2 framework by defining a new type of model block (Figure 6). It is provided with the analog input signals for battery voltage U_{batt} and current I_{batt} and produces values for RUL and

SOC. Prognostics blocks for other system components look similar. The prognostics engine can also be provided with prognosis time t, a planned load profile $\mathcal{P}$, and prognostics model $\mathcal{M}$. We can use the outputs of the prognostics mod-

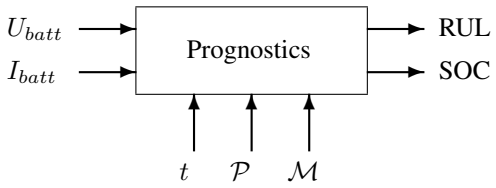


Figure 6. R2U2 Prognostics Block

ule in different ways: a model-based accurate estimate of the battery state at the current time. Because SOC is a statistical variable with a probability density, we use μ_b^{SOC} for the expected value and σ^{SOC} for the standard deviation. These values [IRC: Something is missing here!!], although no predictions give more insight into the actual condition of the battery than just battery voltage.

The RUL prediction itself is calculated via $\Delta k_E(k_p)$. We denote $\mu_b^{RUL}[t, \mathcal{P}, \mathcal{M}]$ for the expected value for the battery's RUL in t time-stamps from now. The load-profile $\mathcal{P}$ can be obtained from the flight computer by reading the list of upcoming way-points. There can be substantial uncertainty in the prediction because of unmodeled and unpredictable external effects like wind. Finally, $\mathcal{M}$ is the battery's prognostic model. For convenience, we introduce $\mu_b^{RUL}[t]$ for a standard expected load profile and μ_b^{RUL} for an RUL prediction at the current time. Those expressions use a nominal battery model.

In the following, we discuss a number of safety and performance properties that actively use the prognosed values of the main battery of the UAS. Most R2U2 health models consist of Boolean and temporal observers to monitor value ranges, relationships, and flight rules. Value checks test whether data values are plausible and relationships encode dependencies among sensor or software data that may originate from different subsystems. Flight rules are defined by institutions (e.g., the Federal Aviation Administration) or are rules that must be obeyed for mission- or system-specific reasons.

3.1. Value Checks

V1: Always have enough battery.

$$\Box(\mu_b^{SOC} > 50)$$

requires that the battery always has a charge of at least 50%[4]. This formula continuously monitors the actual battery state at each point in time during the flight. This threshold will certainly vary depending on the mission profile. Such constraints can be easily formulated in logic. For example, a loaded UAS should have its batteries charged to a minimum of 80%: $is_loaded \rightarrow \Box(\mu_b^{SOC} > 80)$. We might require that the battery will be charged at the end of the flight t_{end} , given the current set of way-points $\mathcal{P}$, and we want to know it now:

$$\mu_b^{RUL}[t_{end}, \mathcal{P}]$$

provides this information. Note that although the formula has a clear temporal meaning, it is not expressed in temporal logic.

V2: The above formula can be refined to monitor for safe operation in specific scenarios: for a safe takeoff and subsequent mission, the battery must be charged to at least 80%

$$\Box(\neg((cmd == takeoff) \wedge (\mu_b^{SOC} < 80))$$

V3: The maximum current that can be drawn from the battery is, of course, always limited: $\Box(I_{batt} < 100A)$. However, an already substantially discharged battery should not be overburdened. For example, for the remaining 10 minutes before the end of RUL, only 30A should be drawn. Several levels of maximum allowable current, depending on the battery's RUL can be specified by:

$$\begin{aligned} &\Box(\mu_b^{RUL} > 100m) \vee I_{batt} < 100A) \quad \vee \\ &\Box(\mu_b^{RUL} > 50m) \vee I_{batt} < 50A) \quad \vee \\ &\Box(\mu_b^{RUL} > 10m) \vee I_{batt} < 30A) \end{aligned}$$

Here again, multiple versions with different battery models might be of interest. A safety rule might restrict the amount of current to be drawn even further based on a battery model that performs predictions of an overheated battery.

V4: don't draw too much current if you are unsure about the charge status of the battery

$$\Box(\neg((\sigma_b^{SOC} > 10) \wedge (I_{eng} > 50A)))$$

V5: Do not heat up the battery as a result of extended use if it is close to empty. If the battery is charged to only 50% or less, a current of more than $30A$ should not be drawn for more than 30 consecutive seconds:

$$\Box((\mu_b^{SOC} < 50) \rightarrow (I_{batt} > 30A)\mathcal{U}_{[0,29s]}(I_{batt} \leq 30A))$$

V6: This property monitors that the landing command is issued before the battery reaches a dangerous low level of 20% or less. Here again, the temporal operators cannot work with

[4]Note that this number has been arbitrarily chosen for illustration purposes.

non-constant intervals, so we discretize into several ranges:

$$\begin{aligned}&\Box(\mu_b^{RUL} < 1000 \rightarrow \Diamond_{[0,1000]}(cmd == landing)) \quad \vee\\ &\Box(\mu_b^{RUL} < 500 \rightarrow \Diamond_{[0,500]}(cmd == landing)) \quad \vee\\ &\Box(\mu_b^{RUL} < 100 \rightarrow \Diamond_{[0,100]}(cmd == landing)) \quad \vee\\ &\ldots\end{aligned}$$

3.2. Relationships

These properties relate signals from different sensors. Here again, rules can be different depending on the state of that battery. For these examples and our simulation experiments, we assume that the throttle value just regulates the motor current I_{eng}, but not engine power or its speed (RPM). With such a simple controller (mostly found in small hobby-style UAS), the motor RPM will decrease if the battery becomes discharged and U_{batt} drops.

R1: Pitching up should result in a strong climb

$$\Box(\Box_{[0,20s]}((\alpha > \alpha_0) \wedge (\mu_b^{SOC} > 80)) \rightarrow \Diamond_{[0,2s]} V_z > 10)$$

For weaker battery the available engine power is lower and thus we cannot expect the same climb speed. We obtain

$$\Box(\Box_{[0,20s]}((\alpha > \alpha_0) \wedge (\mu_b^{SOC} > 30)) \rightarrow \Diamond_{[0,2s]} V_z > 2)$$

3.3. Flight Rules

Flight rules can be complex formulas that must be valid for the UAS to be in a safe and healthy state and to accomplish the mission successfully. They might concern certain minimal performance rules or behavior in specific situations, e.g., a certain loitering altitude should be reached in case the communication link is lost and wait for the communication to be re-established. Some of the flight rules depend on state of the battery. For example, loitering should be attempted only if the battery will have power for at least 5 more minutes of flight. If the battery is too weak, the only safe alternative is to immediately attempt an emergency landing.

F1: This flight rule checks on availability of enough battery power for an upcoming climb in 10min. As discussed in Section 1, we want to be able to decide immediately if the flight rule is violated or not. Figure 7 shows this nominal situation at time t. The blue line corresponds to the future development of the altitude with a climb in 10 minutes. A naive formulation in MTL as $\Box_{[0,10]}(battery == OK)$ is not suitable here (green line), because the formula is false until $t = t+10min$. Only then, the formula can be decided, in our nominal case, to be true. A black triangle depicts the point when the formula is decided. Even the R2U2 synchronous observer only can obtain a "maybe" for the next 10 minutes (red line). By using the prognostics engine, this flight rule can be encoded as

$$(\mu_b^{RUL} > 30) \wedge (t_{climb} - t > 10)$$

where t_{climb} is the planned time of the next climb as obtained from the list of waypoints in the FSW. This formula can be decided right away (magenta in Figure 7). Alternatively, this formula can be written as $\mu_b^{RUL}[10] > 30$.

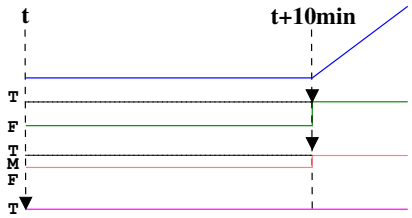


Figure 7. Flight rule **F1**

4. EXPERIMENTS AND RESULTS

4.1. The UAS Testbed

For this paper, we consider a simple and small UAS platform, the NASA Dragon Eye (Figure 8A). With a wingspan of 1.1m it is small, but shares many commonalities with larger and more complex UAS. Our R2U2 framework has been implemented on an Adapteva Parallella board. This credit card sized board features a Zync 7010 FPGA and is running Linux, which facilitates preprocessing of signals, running the prognostics engine, and perform data logging. For our experiments, we also performed the BN reasoning under Linux. Monitoring of the FSW is performed according to the schematics shown in Figure 3. Figure 8B shows the integration of the Parallella board, located in one of the wings.

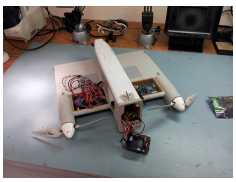
A  B 

Figure 8. A: Dragon Eye UAS. B: Hardware integration of R2U2.

For the experiments presented in this paper, we used a simulation environment that is based on the APM:Plane simulator[5]. As shown in Figure 9 the UAS is controlled by the operator using a ground control station (GCS). The environment, the aircraft dynamics[6], motors, sensors, and the flight computer is simulated in software running on a host PC. Monitored FSW data are collected from the simulator and sent to the Parallella board via a serial interface (FDTI cable), where monitoring and diagnosis is performed. These data are also recorded for replay purposes. Since this simulator does not contain any suitable battery model, we are using a MACCOR battery test stand which is able to simulate the flight path for

[5] `http://plane.ardupilot.com`
[6] JSBSim Flight Dynamics Model `jsbsim.sourceforge.net`

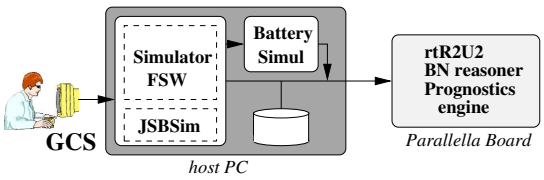


Figure 9. High level system architecture with R2U2

the DragonEye. In the MACCOR we simulate the batteries based on the path scenario in Figure 10. The data collected from the MACCOR testbed is feed to the simulator which in turn runs the prognostics algorithm. In our simplified setting, battery model takes throttle position and returns current values of U_{batt} and I_{batt}.

4.2. Simulation experiments

In this section, we illustrate our approach with some simulation experiments. Figure 10 shows a typical example. The UAS is taking off from altitude 0 and reaching, after a short climb, an altitude of 200ft (Figure 10, top panel). It then flies a level course before descending to 100ft at time stamp 600. A final climb, starting at time stamp 850 brings the UAS back to 200ft altitude. The throttle settings, necessary to execute this profile is proportional to the battery current I_{batt} and shown in Figure 10, 2nd panel from top. The battery voltage drops when a large current is drawn, e.g., during the climb. During the descend, when only very little engine current is drawn, the battery can recover and the voltage slightly increases. Figure 10, 3rd panel from top shows U_{batt} for a fully charged battery (green) and a battery that only had been charged to 50% (red line). With both values, the current and the voltage, the prognostics engine calculates an RUL for the battery at each point in time. As a safety threshold, we use $\theta = 1000$. Finally, the bottom panel shows the output of the R2U2 unit for different formulas related to flight rule **F1**: according to the flight plan, a climb will be necessary at $t = 850$ (dashed magenta line). At time stamp 250, we want to ask if the battery is still good to do that climb. The top row shows the valuation of asynchronous observer for $\Box_{[250,850]}\mu_b^{RUL} > \theta$. This formula can only become true after the interval has been expired. The green line shows this situation for the good battery. The RUL of weak battery, crosses the threshold at around 500 time stamps. After that time, the formula remains false. Thus, the result of this formula can only be used after time stamp 850, usually too late to start any mitigation action. The second line shows the result of the synchronous observer for the same formula. At the beginning its value is "maybe" until the final value can be determined: at $t = 850$ for the good battery and at $t = 500$ for the weak one. Here, this three-valued observer allows us to obtain information earlier. The bottom lines show the valuation of the prognostics-based formula $\mu_b^{RUL}(850) > \theta$ for both scenarios. This formula is valuated immediately and the result can be used to start a necessary mitigation action earlier than with temporal logic alone.

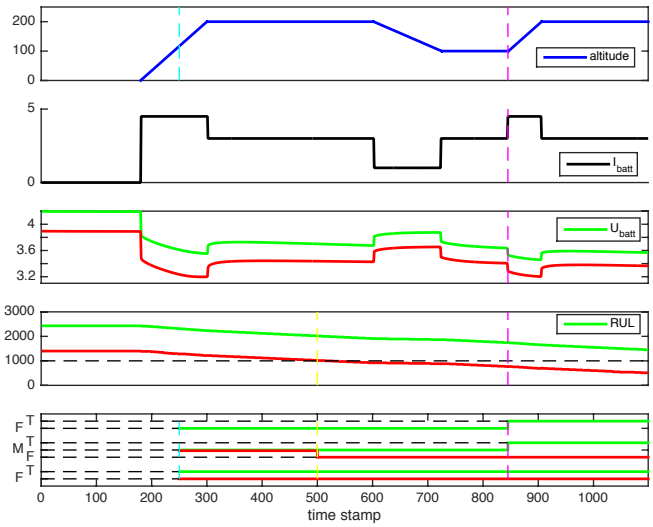


Figure 10. Typical scenario showing illustration of altitude profile, I_{batt}, U_{batt}, μ_b^{RUL}, and valuations (at $t = 250$) of asynchronous and synchronous observers for $\Box_{[250,850]}OK_{batt}$, and $\mu_b^{RUL}(850) > 1000$ for a fully charged battery (green) and one charged to 50% (red).

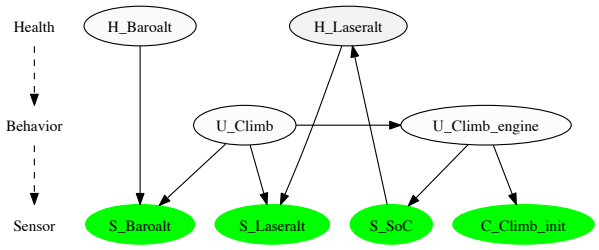


Figure 11. Bayesian Network for altimeter health model

4.3. Prognostics for Root Cause Analysis

For this experiment we assume that the UAS is equipped with a barometric altimeter (BA) and a laser altimeter (LA). Both sensors measure the altitude of the aircraft but can fail in different ways. We want to construct a probabilistic health model to reason about the health of these sensors.

Our BN model in Figure 11 does not reason about actual altitude measurements but rather uses an abstracted indication of increasing or decreasing altitude, noted as UP and DOWN. Observable sensor nodes for BA and LA (Figure 11 bottom, left) are clamped to these values based upon the information extracted from the flight software. The health of each of the sensors is reflected in the nodes H_Laseralt and H_Baroalt with states GOOD and BAD (Figure 11 top).

The priors of these nodes indicate the reliability of each of the sensors (see below). Each of the sensors should mea-

Table 1. CPT Table for BN in Figure 11

A

H_Baroalt	GOOD		BAD	
U_Climb	UP	DOWN	UP	DOWN
S_Baroalt=UP	1	0	0.5	0.5
S_Baroalt=DOWN	0	1	0.5	0.5

B

U_Climb_engine	UP	UP
S_SoC=High	0.85	0.15
S_SoC=Low	0.5	0.5

sure the same entity and thus should show the same behavior with respect to UAS climbs and descents. This is modeled by the unobservable behavior node U_climb, which influences both sensors. If both sensors show the same behavior (e.g., UP,UP), the sensors are most likely healthy. Divergent behavior might indicate a problem with either sensor. Table 1A shows the CPT for BA sensor node S_Baroalt (the CPT table for the LA sensor has the same structure). The CPT table denotes that a healthy sensor follows the climb behavior (center columns). If the sensor is broken, however, probabilities are 0.5, so nothing can be deduced (right columns).

We now need to address the question on how to find the root cause with divergent sensor readings. In our model, we pull in additional information related to climb or descend behavior of the UAS. This behavior is, in particular, related to the question whether an engine-induced climb is happening. This behavior U_Climb_engine is unobservable as well. An engine-induced climb can only happen if a climb command has been initiated by the FSW (C_Climb_init is set to UP). In addition, the current state of the battery plays an important role: if the battery is fully charged (S_SoC is HIGH), a commanded climb will result in the aircraft gaining altitude, something that cannot be guaranteed when the battery is already weak. Here, we need to use probabilistic reasoning, because other effects like down-drafts can play a major role. The CPT for this node is shown in Table 1B. A good battery will result in a climb in 85% of the cases: $p(\text{U\_climb\_engine} = \text{UP}|\text{S\_SoC} = \text{HIGH}) = 0.85$. A weak battery is modeled as $p(\text{U\_climb\_engine} = \text{UP}|\text{S\_SoC} = \text{LOW}) = 0.5$. Note that there is also an edge connecting S_SoC with H_Laseralt. This edge models the effect that the LA might be less reliable and accurate when the battery is weak. In our model, the BA is reliable to 95%, whereas the LA is only 90% reliable if the battery is good. Otherwise, the probability for a healthy LA drops to 0.6.[7]

Figure 12 shows the BN in a number of different situations. Observable nodes are colored red for DOWN and LOW, and green for UP and HIGH, respectively. The marginal probabilities of the behavior and health nodes are shaded in different levels of grey, where white indicates a probability of one. In a nominal scenario (Figure 11) with sensor input consistent to climbing, both sensors appear to be healthy. Figure 12A shows a nominal situation when the UAS is descending. Note that both unobservable behavior nodes now have status DOWN with high probability. A failing barometric altimeter can be detected easily as such during descending (Figure 12B) and climbing (Figure 12C) when the battery is fresh. If we get the same altitude sensor readings, but the battery is weak (Figure 12D), a different picture emerges: the model is aware that commanded climbs with a weak battery often do not result in an increasing altitude, because the engine doesn't receive enough power. In addition, the LA is less reliable with a weak battery. Therefore, the BN reasons that, despite two other disagreeing sources, the BA is most likely in a better shape than the LA.

[7]The probabilities used in this example have been selected for illustration purposes only.

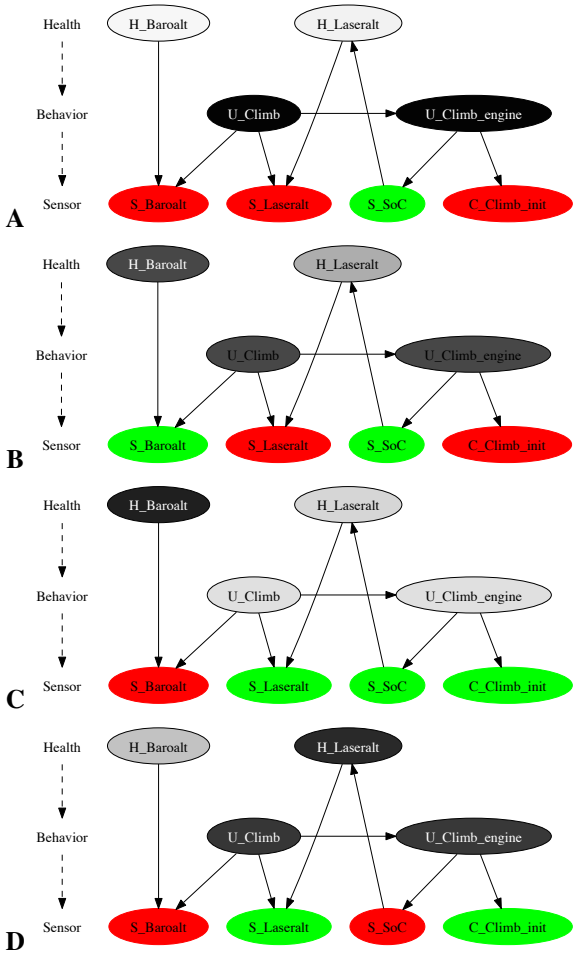


Figure 12. Bayesian Network for altimeter health model in various scenarios. See text for details.

5. CONCLUSIONS

In this paper we presented the integration of prognostics reasoning into the R2U2 temporal and Bayesian health management framework. We used a UKF-based prognostics engine to provide SOC and RUL prognoses for the battery of a UAS.

This information can be used in the R2U2 model to improve diagnostic accuracy. Most notably, prognostic information allows the monitor to instantaneously evaluate at the present time step the properties dealing with events in the future (e.g., a climb in 10 minutes). Temporal logic observers can only do that valuation at the end of the time interval, or only provide minimal information (maybe). Thus, statistically reliable health information is available at a much earlier time, a capability that is very valuable for safe autonomous decision making.

The presented work is only a first step toward full integration. Future work will investigate how the probability densities of the prognostics outputs can be directly used by a Bayesian network with sensor nodes capable of handling discretized probability density functions. Also the use of multiple models for nominal and off-nominal battery conditions and fault progression might improve fault detection and diagnostic reasoning in advanced R2U2 health models. Furthermore, we will aim to implement the prognostics engine in FPGA hardware (see e.g., (Soh & Wu, 2014) for a possible approach). For monitoring safety and performance of a UAS, we would also like to evaluate the usefulness of information from other components of the UAS (e.g. engine, or other structural components) as well as environmental effects (e.g., icing) or operational performance (e.g., reliability of a radio link in different weather conditions) for prognosis. Such additional information might be amenable to prognostic modeling and subsequent integration into R2U2 health models.

NOMENCLATURE

μ_b^{RUL}	mean of RUL for battery
σ_b^{RUL}	std of RUL for battery
μ_b^{SoC}	mean of SoC for battery
σ_b^{SoC}	std of SoC for battery
V_z	vertical velocity
SOC	state of charge of battery [%]
RUL	remaining useful life
EOD	End of Discharge
EOL	End of Life
H_X	BN health node for component X
U	BN behavior node

REFERENCES

Chavira, M., & Darwiche, A. (2005). Compiling Bayesian networks with local structure. In *Proc. 19th IJCAI* (pp. 1306–1312).

Daigle, M., & Goebel, K. (2013). Model-based prognostics with concurrent damage progression processes. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *43*(4), 535-546.

Daigle, M., & Kulkarni, C. (2013). Electrochemistry-based Battery Modeling for Prognostics. In *Proc. PHM 2013* (p. 249-261).

Daigle, M., Kulkarni, C., & Gorospe, G. (2014). Application of model-based prognostics to a pneumatic valves testbed. In *Proc. IEEE aerospace conference.*

Daigle, M., Saha, B., & Goebel, K. (2012). A comparison of filter-based approaches for model-based prognostics. In *Proc. IEEE Aerospace Conference.*

Daigle, M., Saxena, A., & Goebel, K. (2012). An efficient deterministic approach to model-based prediction uncertainty estimation. In *Annual Conf. of the PHM society* (p. 326-335).

Frank, P. M. (1996). Analytical and qualitative model-based fault diagnosis–a survey and some new results. *European Journal of control*, *2*(1), 6–28.

Geist, J., Rozier, K. Y., & Schumann, J. (2014). Runtime observer pairs and Bayesian network reasoners on-board FPGAs: Flight-certifiable system health management for embedded systems. In RV 2014 (Vol. 8734, pp. 215–230). Springer.

Julier, S. J., & Uhlmann, J. K. (1997). A new extension of the Kalman filter to nonlinear systems. In *Proc. 11th int. symp. on aerospace/defense sensing, simulation and controls* (pp. 182–193).

Julier, S. J., & Uhlmann, J. K. (2004). Unscented filtering and nonlinear estimation. *Proc. IEEE*, *92*(3), 401–422.

Karsai, G., Biswas, G., Abdelwahed, S., Mahadevan, N., & Manders, E. (2006). Model-based software tools for integrated vehicle health management. In *Proc. SMC-IT* (p. 8 pp.-442).

Kulkarni, C., Daigle, M., Gorospe, G., & Goebel, G. (2014). Validation of model-based prognostics for pneumatic valves in a demonstration testbed. In *PHM 2014.*

Luo, J., Pattipati, K. R., Qiao, L., & Chigusa, S. (2008). Model-based prognostic techniques applied to a suspension system. *IEEE Trans. on Systems, Man and Cybernetics, Part A*, *38*(5), 1156 -1168.

Mathur, A., Deb, S., & Pattipati, K. (1998). Modeling and real-time diagnostics in TEAMS-RT. In *American control conference* (Vol. 3, p. 1610-1614 vol.3).

Mengshoel, O. J., Chavira, M., Cascio, K., Poll, S., Darwiche, A., & Uckun, S. (2010). Probabilistic model-

based diagnosis: An electrical power system case study. *IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans*, *40*(5), 874–885.

Orchard, M., & Vachtsevanos, G. (2009). A particle filtering approach for on-line fault diagnosis and failure prognosis. *Trans. of the Inst. of Measurement and Control*(3-4), 221-246.

Pearl, J. (1985). A constraint propagation approach to probabilistic reasoning. In *UAI* (pp. 31–42). AUAI Press.

Rahn, C. D., & Wang, C.-Y. (2013). *Battery systems engineering*. Wiley.

Reinbacher, T., Rozier, K. Y., & Schumann, J. (2014). Temporal-logic based runtime observer pairs for system health management of real-time systems. In Proc. TACAS (Vol. 8413, pp. 357–372). Springer.

Schumann, J., Mbaya, T., Mengshoel, O. J., Pipatsrisawat, K., Srivastava, A., Choi, A., & Darwiche, A. (2013). Software health management with Bayesian networks. *Innovations in Systems and Softw. Eng*, *9*(2), 1-22.

Schumann, J., Rozier, K. Y., Reinbacher, T., Mengshoel, O. J., Mbaya, T., & Ippolito, C. (2013). Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. In *PHM 2013.*

Schumann, J., Rozier, K. Y., Reinbacher, T., Mengshoel, O. J., Mbaya, T., & Ippolito, C. (2015). Towards real-time, on-board, hardware-supported sensor and software health management for unmanned aerial systems. *IJPHM*, *6*(2), 021.

Soh, J., & Wu, X. (2014). A modular FPGA-based implementation of the Unscented Kalman filter. In *Proc. Adaptive Hardware and Systems (AHS), 2014* (p. 127-134).

Teubert, C., & Daigle, M. (2013). I/P transducer application of model-based wear detection and estimation using steady state conditions. In *PHM 2013* (p. 134-140).

Teubert, C., & Daigle, M. (2014). Current/pressure transducer application of model-based prognostics using steady state conditions. In *2014 IEEE Aerospace.*

BIOGRAPHIES

Dr. Johann Schumann is Chief Scientist for Computational Sciences with SGT, Inc. and working at the NASA Ames Research Center. He received his PhD (1991) and German habilitation degree (2000) in Computer Science from the Technische University Munich in Germany. Dr. Johann Schumann is engaged in research on software health management, verification and validation of IVHM algorithms, analysis and V&V of advanced air traffic control algorithms, and the automatic generation of reliable code. He is author of a book on theorem proving in software engineering and has published numerous articles on automated deduction and its applications, automatic program generation, V&V of safety-critical systems, and neural network oriented topics.

Indranil Roychoudhury received the B.E. (Hons.) degree in Electrical and Electronics Engineering from Birla Institute of Technology and Science, Pilani, Rajasthan, India in 2004, and the M.S. and Ph.D. degrees in Computer Science from Vanderbilt University, Nashville, Tennessee, USA, in 2006 and 2009, respectively. Since August 2009, he has been with SGT, Inc., at NASA Ames Research Center as a Computer Scientist. His research interests include hybrid systems modeling, model-based diagnostics and prognostics, distributed diagnostics and prognostics, and Bayesian diagnostics of complex physical systems. Dr. Roychoudhury is a member of the Prognostics and Health Management Society and a Senior Member of the IEEE.

Chetan S. Kulkarni received the B.E. (Bachelor of Engineering) degree in Electronics and Electrical Engineering from University of Pune, India in 2002 and the M.S. and Ph.D. degrees in Electrical Engineering from Vanderbilt University, Nashville, TN, in 2009 and 2013, respectively. He was a Senior Project Engineer with Honeywell Automation India Limited (HAIL) from 2003 till April 2006. From May 2006 to August 2007 he was a Research Fellow at the Indian Institute of Technology (IIT) Bombay with the Department of Electrical Engineering. From Aug 2007 to Dec 2012, he was a Graduate Research Assistant with the Institute for Software Integrated Systems and Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN. Since Jan 2013 he has been a Staff Researcher with SGT Inc. at the Prognostics Center of Excellence, NASA Ames Research Center. His current research interests include physics-based modeling, model-based diagnosis and prognosis. Dr. Kulkarni is a member of the Prognostics and Health Management (PHM) Society, AIAA and Senior member IEEE.