Advanced Methods for Determining Prediction Uncertainty in Model-Based Prognostics with Application to Planetary Rovers

Matthew Daigle[1] and Shankar Sankararaman[2]

[1] *NASA Ames Research Center, Moffett Field, CA, 94035, USA*
matthew.j.daigle@nasa.gov

[2] *SGT, Inc., NASA Ames Research Center, Moffett Field, CA, 94035, USA*
shankar.sankararaman@nasa.gov

ABSTRACT

Prognostics is centered on predicting the time of and time until adverse events in components, subsystems, and systems. It typically involves both a state estimation phase, in which the current health state of a system is identified, and a prediction phase, in which the state is projected forward in time. Since prognostics is mainly a prediction problem, prognostic approaches cannot avoid uncertainty, which arises due to several sources. Prognostics algorithms must both characterize this uncertainty and incorporate it into the predictions so that informed decisions can be made about the system. In this paper, we describe three methods to solve these problems, including Monte Carlo-, unscented transform-, and first-order reliability-based methods. Using a planetary rover as a case study, we demonstrate and compare the different methods in simulation for battery end-of-discharge prediction.

1. INTRODUCTION

Prognostics focuses on predicting the time of and time until adverse events in components, subsystems, and systems. Model-based methods consist of an estimation phase, in which the current health state of a system is identified, followed by a prediction phase, in which the state is simulated until the adverse event. Because prognostics is mainly a prediction problem, uncertainty, which manifests due to many sources, cannot be avoided. This uncertainty will determine how the system evolves in time, i.e., the system evolution is a random process. To approach this problem in a systematic way, there are two problems to solve: (*i*) characterizing the sources of uncertainty, and (*ii*) quantifying the combined effect of the different sources of uncertainty within the prediction. With proper estimation of the prediction uncertainty, predictions can then be used to make informed decisions about the system.

In many applications, the largest source of uncertainty is that associated with the future inputs, and so most work in uncertainty quantification for prognostics has focused in that area. In the context of fatigue damage prognosis, different types of methods (Ling et al., 2011) such as rainflow counting techniques, auto-regressive moving-average models, and Markov processes have been used for constructing future loading trajectories. In (Sankararaman, Ling, Shantz, & Mahadevan, 2011), the authors construct future input trajectories as sequential blocks of constant-amplitude loading, and such trajectories are sampled in the prediction algorithm. In (Saha, Quach, & Goebel, 2012) the authors characterize the future inputs by determining statistics of the battery loading for typical unmanned aerial vehicle maneuvers based on past flight data, and construct future input trajectories as constrained sequences of flight maneuvers. Constant loading is assumed in (Luo, Pattipati, Qiao, & Chigusa, 2008) for a vehicle suspension system, and predictions are made for a weighted set of three different loading values.

Once each source of uncertainty has been characterized, it must be accounted for within the prediction, and thereby their combined effect on the overall prediction must be quantified. In previous work (Daigle, Saxena, & Goebel, 2012), we investigated methods for accouting for future input uncertainty in the predictions and introduced the unscented transform (UT) method for efficiently estimating the future input uncertainty, however, methods for future input characterization were not discussed and only constant-amplitude loading was considered. In other work (Sankararaman, Daigle, Saxena, & Goebel, 2013; Sankararaman & Goebel, 2013), we investigated the use of analytical methods, namely, first-order reliability (FORM) based methods for propagating the future input uncertainty, however it also was limited to constant-amplitude loading, and the methods were applied only for offline analysis and not within online prognostic algorithms.

In this paper, we adopt a model-based prognostics approach (Orchard & Vachtsevanos, 2009; Daigle & Goebel, 2013; Luo et al., 2008). We discuss the uncertainty representation and quantification problem, and develop a novel generalized framework using the notion of surrogate variables, allowing the representation of state uncertainty, parameter uncertainty, future input uncertainty, and process noise in a common framework that allows consideration of both constant- and variable-amplitude loading situations. We explore three methods for prediction with uncertainty quantification, namely, Monte Carlo sampling, UT sampling, and inverse FORM. Using a rover battery system as a case study, we describe two methods for future input uncertainty characterization, implement the prediction methods, and compare their performance for battery end-of-discharge prediction in simulated constant- and variable-amplitude loading scenarios.

The paper is organized as follows. In Section 2, we define the prognostics problem and develop the uncertainty representation framework. In Section 3, we describe methods for handling uncertainty within the prediction process. In Section 4, we introduce the rover case study and present several examples in simulation demonstrating the methods and comparing their performance. Section 5 concludes the paper.

2. Model-based Prognostics

We first formulate the prognostics problem, and develop the uncertainty representation framework. We then provide an architecture for model-based prognostics.

2.1. Problem Formulation

We assume the system model may be generally defined as

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

$$\mathbf{y}(k) = \mathbf{h}(k, \mathbf{x}(k), \boldsymbol{\theta}(k), \mathbf{u}(k), \mathbf{n}(k)), \tag{2}$$

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.[1] The unknown parameter vector $\boldsymbol{\theta}(k)$ is used to capture explicit model parameters whose values are unknown and time-varying stochastically.

In prognostics, we are interested in predicting the occurrence of some (desirable or undesirable) 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

$$k_E(k_P) \triangleq \inf\{k \in \mathbb{N} \colon k \geq k_P \wedge T_E(\mathbf{x}(k), \boldsymbol{\theta}(k), \mathbf{u}(k)) = 1\}. \tag{3}$$

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

$$\Delta k_E(k_P) \triangleq k_E(k_P) - k_P. \tag{4}$$

In the context of systems health management, the event E corresponds to some undesirable event such as the failure of a system, some process variable out-of-range, or a similar type of situation. 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). In this case, k_E is then conventionally called end of life, while Δk_E is conventionally called remaining useful life.

The system evolution is a random process due to the process noise $\mathbf{v}(k)$ and nondeterministic inputs $\mathbf{u}(k)$. Since the system evolution is a random process, at time k_P, the system takes some trajectory out of many possible trajectories, therefore, k_E and Δk_E are random variables. So, the prognostics problem is to predict the *probability distribution* of k_E (Daigle, Saxena, & Goebel, 2012; Sankararaman et al., 2013).

Problem (Prognostics)**.** The prognostics problem is to, at prediction time k_P, compute $p(k_E(k_P)|\mathbf{y}(k_0{:}k_P))$ and/or $p(\Delta k_E(k_P)|\mathbf{y}(k_0{:}k_P))$.

In practice, obtaining this distribution is difficult because the state at k_P, system model, process noise distribution, and future input distribution may not be known precisely.

2.2. Representing Uncertainty

In order to predict k_E, four sources of uncertainty must be dealt with: (*i*) the initial state at time k_P, $\mathbf{x}(k_p)$; (*ii*) the parameter values $\boldsymbol{\theta}(k)$ for all $k \geq k_P$, denoted as $\boldsymbol{\Theta}_{k_P}$ (the subscript k_P indicates the start time of the trajectory); (*iii*) the inputs $\mathbf{u}(k)$ for all $k \geq k_P$, denoted as $\mathbf{U}_{k_P}$; and (*iv*) the process noise $\mathbf{v}(k)$ for all $k \geq k_P$, denoted as $\mathbf{V}_{k_P}$. In order to make a prediction that accounts for this uncertainty, we require the probability distributions $p(\mathbf{x})$, $p(\boldsymbol{\Theta}_{k_P})$, $p(\mathbf{U}_{k_P})$, and $p(\mathbf{V}_{k_P})$.

For describing the probability distribution of a generic trajectory $\mathbf{A}_k$, we introduce a set of *surrogate* random variables $\boldsymbol{\lambda}_a = [\lambda_a^1 \lambda_a^2 \ldots]$. We describe a trajectory using $\boldsymbol{\lambda}_a$ and instead define $p(\boldsymbol{\lambda}_a)$, which in turn defines $p(\mathbf{A}_k)$. These surrogate variables can be used to describe trajectories in a variety of ways. For example, we can associate $\mathbf{A}_k(k)$ with λ_a^1, $\mathbf{A}_k(k+1)$ with λ_a^2, etc. Or, we can describe a trajectory with a parameterized function, where the parameters are the random variables, e.g., $\mathbf{A}_k(k) = \lambda_a^1 + \lambda_a^2 \sin k$. The use of the surrogate variables provides flexibility to the user in defining the distribution of trajectories. So, for the parameter, input,

[1] Here, we use bold typeface to denote vectors, and use n_a to denote the length of a vector $\mathbf{a}$.

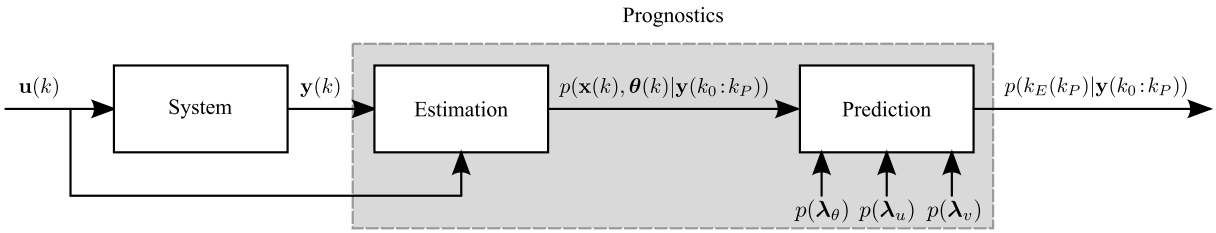


Figure 1. Prognostics architecture.

and process noise trajectories we introduce the surrogate variables $\boldsymbol{\lambda}_\theta$, $\boldsymbol{\lambda}_u$, and $\boldsymbol{\lambda}_v$.

In the model-based prognostics paradigm, the probability distribution for the initial state at the time of prediction k_P, $p(\mathbf{x}(k_P), \boldsymbol{\theta}(k_P))$, is specified by an estimator, such as the Kalman filter, unscented Kalman filter (UKF) (Julier & Uhlmann, 1997), or particle filter (Arulampalam, Maskell, Gordon, & Clapp, 2002). The distribution may be represented by a set of statistical moments like a mean vector and covariance matrix (as in the Kalman filter), or a set of weighted samples (as in the UKF and particle filter). This defines only the parameter vector at k_P and not for future time steps. For the future values of $\boldsymbol{\theta}$, these are drawn from a distribution specified by $\boldsymbol{\lambda}_\theta$. In the simplest case, λ_θ^1 can define $\boldsymbol{\theta}(k_P)$, λ_θ^2 can define $\boldsymbol{\theta}(k_P + 1)$, etc, where the distribution of each λ_θ^i is the same and specified by $p(\boldsymbol{\theta}(k_p))$ as determined by the estimator. Alternatively, it can be assumed that $\boldsymbol{\theta}(k)$ is constant, in which case only one surrogate variable is required, with the distribution specified by the estimator.

For process noise, we define $p(\mathbf{V}_k)$ using $\boldsymbol{\lambda}_v$. It is often assumed in practice that, at each time instant, there is a single probability distribution from which the process noise values are drawn. The distribution is defined a priori based on the understanding of the system and its model. In the simplest case, there is a random variable for every time step, i.e, λ_v^1 defines $\mathbf{v}(k_P)$, λ_v^2 defines $\mathbf{v}(k_P + 1)$, etc. Since the number of random variables depends on the number of time steps, such a treatment potentially leads to dimensionality issues. Sankararaman and Goebel (Sankararaman & Goebel, 2013) have demonstrated that it is possible to approximate the effect of this process noise using an equivalent time-invariant process noise, i.e., a single random variable that defines a constant value of process noise for all k. In this case, $\boldsymbol{\lambda}_v$ would contain only that single random variable, whose probability distribution will be constructed by matching the likelihood of occurrence of the original time-varying process noise.

For the future input trajectories, the distribution depends on the particular system being prognosed and the environment it is operating within. As with the other trajectories, we describe $p(\mathbf{U}_{k_P})$ using $\boldsymbol{\lambda}_u$. Often, there is some knowledge about what the future input will be and only a few random variables are needed in $\boldsymbol{\lambda}_u$. For example, in a constant-loading scenario, the inputs can be defined with $\mathbf{u}(k) = \lambda_u^1$ for $k \geq k_P$. Any arbitrary function parameterized by a set of random variables may be used to define $\mathbf{u}(k)$, e.g., $\mathbf{u}(k) = \lambda_u^1 \cdot sin(k)$, or $\mathbf{u}(k) = \lambda_u^1 + \lambda_u^2 \cdot k$. The variables in $\boldsymbol{\lambda}_\theta$ may or may not be independent.

To predict k_E at time k_P, we require the initial state, $\mathbf{x}(k_P)$; the parameter trajectory up to k_E, $\boldsymbol{\Theta}_{k_P} = [\boldsymbol{\theta}(k_P), \ldots, \boldsymbol{\theta}(k_E)]$; the process noise trajectory up to k_E, $\mathbf{V}_{k_P} = [\mathbf{v}(k_P), \ldots, \mathbf{v}(k_E)]$; and the input trajectory up to k_E, $\mathbf{U}_{k_P} = [\mathbf{u}(k_P), \ldots, \mathbf{u}(k_E)]$. With this information, k_E can be computed simply by simulating the state forward in time, using the state equation $\mathbf{f}$, until T_E evaluates to 1, at which point the current time is k_E. Because we have only probability distributions, we need to propagate the uncertainty through this procedure in order to obtain the distribution for k_E (Sankararaman et al., 2013; Sankararaman & Goebel, 2013). Methods to do such uncertainty propagation will be described in Section 3.

2.3. Prognostics Architecture

We adopt a model-based prognostics architecture (Daigle & Goebel, 2013), in which there are two sequential problems, (*i*) the *estimation* problem, which requires determining a joint state-parameter estimate $p(\mathbf{x}(k), \boldsymbol{\theta}(k)|\mathbf{y}(k_0{:}k_P))$ based on the history of observations up to time k, $\mathbf{y}(k_0{:}k_P)$, and (*ii*) the *prediction* problem, which determines at k_P, using $p(\mathbf{x}(k), \boldsymbol{\theta}(k)|\mathbf{y}(k_0{:}k_P))$, $p(\boldsymbol{\lambda}_\theta)$, $p(\boldsymbol{\lambda}_u)$, and $p(\boldsymbol{\lambda}_v)$, 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 from $k_E(k_P)$.

The prognostics architecture is shown in Fig. 1. 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 the distributions for the surrogate variables, $p(\boldsymbol{\lambda}_\theta)$, $p(\boldsymbol{\lambda}_u)$, and $p(\boldsymbol{\lambda}_v)$, to compute the probability distribution $p(k_E(k_P)|\mathbf{y}(k_0{:}k_P))$ at given prediction times k_P.

Algorithm 1 $k_E(k_P) \leftarrow \texttt{P}(\mathbf{x}(k_P), \boldsymbol{\Theta}_{k_P}, \mathbf{U}_{k_P}, \mathbf{V}_{k_P})$

$k \leftarrow k_P$
$\mathbf{x}(k) \leftarrow \mathbf{x}(k_P)$
while $T_E(\mathbf{x}(k), \boldsymbol{\Theta}_{k_P}(k), \mathbf{U}_{k_P}(k)) = 0$ **do**
  $\mathbf{x}(k+1) \leftarrow \mathbf{f}(k, \mathbf{x}(k), \boldsymbol{\Theta}_{k_P}(k), \mathbf{U}_{k_P}(k), \mathbf{V}_{k_P}(k))$
  $k \leftarrow k+1$
  $\mathbf{x}(k) \leftarrow \mathbf{x}(k+1)$
end while
$k_E(k_P) \leftarrow k$

3. PREDICTION

Prediction is initiated at a given time k_P using the current joint state-parameter estimate, $p(\mathbf{x}(k_P), \boldsymbol{\theta}(k_P)|\mathbf{y}(k_0{:}k_P))$. Approaches to determine this estimate are reviewed in (Daigle, Saha, & Goebel, 2012) and are not described here. The goal is to compute $p(k_E(k_P)|\mathbf{y}(k_0{:}k_P))$ using the state-parameter estimates and assumptions about uncertainty regarding the future parameter, input, and process noise values.

Consider one realization of each of the uncertain quantities at prediction time k_P: the state $\mathbf{x}(k_P)$, the parameter trajectory $\boldsymbol{\Theta}_{k_P}$, the input trajectory $\mathbf{U}_{k_P}$, and the process noise trajectory $\mathbf{V}_{k_P}$. Then, the corresponding realization of k_E can be computed with the system model as shown in Algorithm 1. In Algorithm 1, the function P simulates the system model until the threshold T_E evaluates to 1.

This algorithm requires computing first realizations of the state-parameter distribution, the parameter trajectory, the input trajectory, and the process noise trajectory. As described in Section 2, the distribution for the state comes from an estimator, and the distributions for the parameter, input, and process noise trajectories are defined indirectly by a set of surrogate variables. At the higher level, we are interested in computing the distribution for k_E from the distributions for $p(\mathbf{x}(k_P), \boldsymbol{\theta}(k_P))$, $p(\boldsymbol{\lambda}_\theta)$, $p(\boldsymbol{\lambda}_u)$, and $p(\boldsymbol{\lambda}_v)$.

The function that takes these surrogate variables and computes a distribution for k_E, which we refer to as G, is the real function we are interested in, i.e., $p(k_E) = \texttt{G}(p(\mathbf{x}(k_P)), p(\boldsymbol{\theta}(k_P)), p(\boldsymbol{\lambda}_\theta), p(\boldsymbol{\lambda}_u), p(\boldsymbol{\lambda}_v))$. To compute $p(k_E)$, we must propagate the uncertainty through this function. That is, predicting $p(k_E(k_P)|\mathbf{y}(k_0{:}k_P))$ is an uncertainty propagation problem. In the following subsections, we describe three different methods with which to solve this problem. They each compute realizations of the state, parameter trajectory, input trajectory, and process noise trajectory, and call the P function to obtain a realization of k_E. They differ in how they compute these realizations and how they construct $p(k_E(k_P)|\mathbf{y}(k_0{:}k_P))$ from them, and, consequently, in their computational complexity.

3.1. Monte Carlo Sampling

To account for uncertainty in the prediction step, the simplest method is Monte Carlo sampling. Several realizations of the state, parameter trajectory, input trajectory, and process noise trajectory are sampled from their corresponding distributions. For each realization, k_E is computed. The resulting set of k_E values characterizes its distribution.

Algorithm 2 $\{k_E^{(i)}\}_{i=1}^N = \texttt{MC}(p(\mathbf{x}(k_P), \boldsymbol{\theta}(k_P)|\mathbf{y}(k_0{:}k_P)), p(\boldsymbol{\lambda}_\theta), p(\boldsymbol{\lambda}_u), p(\boldsymbol{\lambda}_v), N)$

for $i = 1$ **to** N **do**
  $(\mathbf{x}^{(i)}(k_P), \boldsymbol{\theta}^{(i)}(k_P)) \sim p(\mathbf{x}(k_P), \boldsymbol{\theta}(k_P)|\mathbf{y}(k_0{:}k_P))$
  $\boldsymbol{\lambda}_\theta^{(i)} \sim p(\boldsymbol{\lambda}_\theta)$
  $\boldsymbol{\Theta}_{k_P}^{(i)} \leftarrow \texttt{construct}\Theta(\boldsymbol{\lambda}_\theta^{(i)}, \boldsymbol{\theta}^{(i)}(k_P))$
  $\boldsymbol{\lambda}_u^{(i)} \sim p(\boldsymbol{\lambda}_u)$
  $\mathbf{U}_{k_P}^{(i)} \leftarrow \texttt{constructU}(\boldsymbol{\lambda}_u^{(i)})$
  $\boldsymbol{\lambda}_v^{(i)} \sim p(\boldsymbol{\lambda}_v)$
  $\mathbf{V}_{k_P}^{(i)} \leftarrow \texttt{constructV}(\boldsymbol{\lambda}_v^{(i)})$
  $k_E^{(i)} \leftarrow \texttt{P}(\mathbf{x}^{(i)}(k_P), \boldsymbol{\Theta}_{k_P}^{(i)}, \mathbf{U}_{k_P}^{(i)}, \mathbf{V}_{k_P}^{(i)})$
end for

Algorithm 2 shows the Monte Carlo prediction algorithm. The algorithm is given as input the joint state-parameter distribution, and the distributions of the $\boldsymbol{\lambda}_\theta$, $\boldsymbol{\lambda}_u$, and $\boldsymbol{\lambda}_v$ variables, along with the number of samples to take, N. For N times, the algorithm samples from the distributions, constructs the parameter, input, and process noise trajectories, and calls the P function to compute k_E. To construct the trajectories from the $\boldsymbol{\lambda}$ variables, the constructΘ constructU and constructV functions must be provided by the user, as they depend on the chosen surrogate variables and how they are to be interpreted. Note that the constructΘ function requires an additional input, $\boldsymbol{\theta}^{(i)}(k_P)$, which is a sample from the estimator-computed joint parameter estimate at time k_P.

In any prediction algorithm, computational complexity is driven by two factors: the number of evaluations of P, and the length of time each sample takes to simulate to k_E (Daigle & Goebel, 2010). Assuming a fair comparison for the second factor, we can compare algorithms mainly by the first factor. In the case of Monte Carlo sampling, the number of samples N is arbitrary and determines the efficiency. In most cases, a very large value of N is required in order to reproduce accurately the important characteristics of the k_E distribution.

3.2. Unscented Transform Sampling

A more complex method that can improve the efficiency of prediction over the Monte Carlo method is to use the unscented transform (UT) to sample from the distributions (Daigle, Saxena, & Goebel, 2012). We present here an extended and generalized version of the method originally presented in (Daigle, Saxena, & Goebel, 2012) in order to accommodate the $\boldsymbol{\lambda}$ variable formulation.

The UT takes a random variable $\mathbf{a} \in \mathbb{R}^{n_a}$, with mean $\bar{\mathbf{a}}$ and covariance $\mathbf{P}_{aa}$, that is related to a second random variable

$\mathbf{b} \in \mathbb{R}^{n_b}$ by some function $\mathbf{b} = \mathbf{g}(\mathbf{a})$, and computes the mean $\bar{\mathbf{b}}$ and covariance $\mathbf{P}_{bb}$ with high accuracy using a minimal set of deterministically selected weighted samples, called *sigma points* (Julier & Uhlmann, 1997). The number of sigma points is linear in the dimension of the random variable, and so the statistics (mean and covariance) of the transformed random variable can be computed more efficiently than Monte Carlo sampling.

Here, $\boldsymbol{\mathcal{A}}^i$ denotes the ith sigma point from $\mathbf{a}$ and w^i denotes its weight. The sigma points are always chosen such that the mean and covariance match those of the original distribution, $\bar{\mathbf{a}}$ and $\mathbf{P}_{aa}$. Each sigma point is passed through $\mathbf{g}$ to obtain new sigma points $\boldsymbol{\mathcal{B}}$, i.e.,

$$\boldsymbol{\mathcal{B}}^i = \mathbf{g}(\boldsymbol{\mathcal{A}}^i)$$

with mean and covariance calculated as

$$\bar{\mathbf{b}} = \sum_i w^i \boldsymbol{\mathcal{B}}^i$$

$$\mathbf{P}_{bb} = \sum_i w^i (\boldsymbol{\mathcal{B}}^i - \bar{\mathbf{b}})(\boldsymbol{\mathcal{B}}^i - \bar{\mathbf{b}})^T.$$

In this paper, we use the symmetric unscented transform (SUT), in which $2n_a + 1$ sigma points are symmetrically selected about the mean according to (Julier & Uhlmann, 2004):

$$w^i = \begin{cases} \dfrac{\kappa}{(n_a + \kappa)}, & i = 0 \\ \dfrac{1}{2(n_a + \kappa)}, & i = 1, \ldots, 2n_a \end{cases}$$

$$\boldsymbol{\mathcal{A}}^i = \begin{cases} \bar{\mathbf{a}}, & i = 0 \\ \bar{\mathbf{a}} + \left(\sqrt{(n_a+\kappa)\mathbf{P}_{aa}}\right)^i, & i = 1, \ldots, n_a \\ \bar{\mathbf{a}} - \left(\sqrt{(n_a+\kappa)\mathbf{P}_{aa}}\right)^i, & i = n_a+1, \ldots, 2n_a, \end{cases}$$

where $\left(\sqrt{(n_a + \kappa)\mathbf{P}_{aa}}\right)^i$ refers to the ith column of the matrix square root of $(n_a + \kappa)\mathbf{P}_{aa}$, and κ is a free parameter that can be used to tune higher order moments of the distribution. When $\mathbf{a}$ is assumed to be Gaussian, (Julier & Uhlmann, 1997) recommends setting $\kappa = 3 - n_a$. Note that with the UT, weights may be negative, and are not to be directly interpreted as probabilities.

For prediction, the G function serves as $\mathbf{g}$ in the above formulation, where $\mathbf{a}$ corresponds to the joint distribution of the state and $\boldsymbol{\lambda}$ variables, and $\mathbf{b}$ corresponds to k_E. The prediction algorithm in this case is shown as Algorithm 3. The algorithm first uses the symmetric unscented transform to compute sigma points for the given probability distributions (treated together as a joint distribution), where each sigma point consists of a sample for the state-parameter vector and the $\boldsymbol{\lambda}$ variables. For each sigma point, the parameter, input, and process noise trajectories are constructed and the P function is called to compute the corresponding k_E. The mean and variance for k_E can be computed from its sigma points.

Algorithm 3 $\{k_E^{(i)}, w^{(i)}\}_{i=1}^N = \texttt{UT}(p(\mathbf{x}(k_P), \boldsymbol{\theta}(k_P)|\mathbf{y}(k_0{:}k_P)), p(\boldsymbol{\lambda}_\theta), p(\boldsymbol{\lambda}_u), p(\boldsymbol{\lambda}_v))$

$N \leftarrow 2(n_x + n_\theta + n_{\lambda_\theta} + n_{\lambda_u} + n_{\lambda_v}) + 1$
$\{\mathbf{x}^{(i)}(k_P), \boldsymbol{\theta}^{(i)}(k_P), \boldsymbol{\lambda}_\theta, \boldsymbol{\lambda}_u, \boldsymbol{\lambda}_v, w^{(i)}\}_{i=1}^N \leftarrow \texttt{SUT}((p(\mathbf{x}(k_P), \boldsymbol{\theta}(k_P)|\mathbf{y}(k_0{:}k_P)), p(\boldsymbol{\lambda}_\theta), p(\boldsymbol{\lambda}_u), p(\boldsymbol{\lambda}_v)))$
for $i = 1$ **to** N **do**
  $\boldsymbol{\Theta}_{k_P}^{(i)} \leftarrow \texttt{constructΘ}(\boldsymbol{\lambda}_\theta^{(i)}, \boldsymbol{\theta}^{(i)}(k_P))$
  $\mathbf{U}_{k_P}^{(i)} \leftarrow \texttt{constructU}(\boldsymbol{\lambda}_u^{(i)})$
  $\mathbf{V}_{k_P}^{(i)} \leftarrow \texttt{constructV}(\boldsymbol{\lambda}_v^{(i)})$
  $k_E^{(i)} \leftarrow \texttt{P}(\mathbf{x}^{(i)}(k_P), \boldsymbol{\Theta}_{k_P}^{(i)}, \mathbf{U}_{k_P}^{(i)}, \mathbf{V}_{k_P}^{(i)})$
end for

This prediction method will often require a smaller number of samples than with Monte Carlo sampling, since the number of sigma points grows only linearly with the problem dimension. This is partly due to the fact that the UT method provides only mean and covariance information, whereas additional higher-order moments can be computed with the Monte Carlo method. Extended versions of the UT are also available that compute higher-order statistical moments (Julier, 1998).

3.3. Inverse First-Order Reliability Method

The Monte Carlo and UT approaches are sampling-based techniques to predict the uncertainty in the k_E. Here we briefly explain an *optimization-based* method, the inverse first-order reliability method, for this purpose. The First-order Reliability Method (FORM) and the Inverse First-Order Reliability Method (inverse FORM) were originally developed by structural engineers to evaluate the probability of failure of a given structure (Haldar & Mahadevan, 2000). In an earlier publication (Sankararaman et al., 2013), we have extended these two approaches for uncertainty quantification in the context of remaining useful life estimation, i.e., propagate the uncertainty in present estimates of states and parameters, future loading, future process noise, and future parameter values through P (defined earlier in Algorithm 1) to calculate the uncertainty in k_E. In the present paper, we use the inverse FORM methodology to calculate the entire probability distribution of k_E in terms of the cumulative distribution function. Calculating the cumulative distribution function (CDF) is equivalent to calculating the probability density function $p(k_E(k_P)|\mathbf{y}(k_0{:}k_P))$, since the density function can easily be obtained by differentiating the cumulative distribution function.

For a generic random variable Z, the cumulative distribution function is a mapping from a realization z of the random variable to the set $[0, 1]$, and is denoted by $F_Z(z)$. If $F_Z(z) = \eta$, then the probability that the random variable Z is less than a given value z is equal to η. In the context of prognostics, the goal is to compute the uncertainty in k_E. Typically, the

Algorithm 4 $k_E(k_P) \leftarrow \mathtt{P}_{\boldsymbol{\lambda}}(\omega)$

$[\mathbf{x}(k_P), \boldsymbol{\theta}(k_P), \boldsymbol{\lambda}_\theta, \boldsymbol{\lambda}_u, \boldsymbol{\lambda}_v] \leftarrow \omega$
$\boldsymbol{\Theta}_{k_P} \leftarrow \mathtt{construct}\Theta(\boldsymbol{\lambda}_\theta, \boldsymbol{\theta}(k_P))$
$\mathbf{U}_{k_P} \leftarrow \mathtt{constructU}(\boldsymbol{\lambda}_u)$
$\mathbf{V}_{k_P} \leftarrow \mathtt{constructV}(\boldsymbol{\lambda}_v)$
$k_E(k_P) \leftarrow \mathtt{P}(\mathbf{x}(k_P), \boldsymbol{\Theta}_{k_P}, \mathbf{U}_{k_P}, \mathbf{V}_{k_P})$

quantities $\mathbf{x}(k_P)$, $\boldsymbol{\theta}(k_P)$, $\boldsymbol{\lambda}_\theta$, $\boldsymbol{\lambda}_u$, and $\boldsymbol{\lambda}_v$ are vectors, and now, consider a new vector which is the concatenation of all these vectors as $\omega = [\mathbf{x}(k_P), \boldsymbol{\theta}(k_P), \boldsymbol{\lambda}_\theta, \boldsymbol{\lambda}_u, \boldsymbol{\lambda}_v]$. Based on the probability distributions of $\mathbf{x}(k_P)$, $\boldsymbol{\theta}(k_P)$ $\boldsymbol{\lambda}_\theta$, $\boldsymbol{\lambda}_u$, $\boldsymbol{\lambda}_v$, the joint probability density of ω, denoted as $f_{\boldsymbol{\Omega}}(\omega)$, can be easily calculated. Note that ω is a realization of the random variable that is denoted by $\boldsymbol{\Omega}$.

In order to implement the inverse FORM, it is necessary to construct a function whose inputs are $\omega = [\mathbf{x}(k_P), \boldsymbol{\theta}(k_P), \boldsymbol{\lambda}_\theta, \boldsymbol{\lambda}_u, \boldsymbol{\lambda}_v]$ and the output is k_E. This function similar to $\mathtt{P}$, with one difference; while $\mathtt{P}$ takes realizations of entire trajectories, i.e., $\boldsymbol{\Theta}_{k_P}$,$\mathbf{U}_{k_P}$, and $\mathbf{V}_{k_P}$ as input arguments, the new function needs to consider realizations of the corresponding surrogate variables as input arguments. This new function, denoted by $\mathtt{P}_\lambda$, is indicated in Algorithm 4.

The inverse FORM approach is now explained using the function $k_E = \mathtt{P}_{\boldsymbol{\lambda}}(\omega)$. The reason for such vectorized representation using ω is not only to facilitate easy explanation of the FORM and inverse FORM algorithms but also demonstrate that these algorithms do not differentiate amongst state estimate values, parameter values, future loading trajectories, and process noise trajectories but treat them similarly.

Let the CDF of k_E be denoted as $F_{K_e}(k_E) = \eta$. Using $k_E = \mathtt{P}_{\boldsymbol{\lambda}}(\omega)$, the FORM approach can be used to calculate the value of η corresponding to a given value of k_E. Conversely, the inverse FORM approach can be used to calculate the value of k_E corresponding to a given value of η. By repeating the FORM procedure for multiple values of k_E, or by repeating the inverse FORM procedure for multiple values of η, the entire CDF $F_{K_e}(k_E)$ can be calculated. In a practical scenario, it is not reasonable to know what values of k_E need to be selected to implement the FORM procedure, since the goal is actually to compute the uncertainty in k_E. However, it is easier to select values of η (say, 0.1, 0.2, 0.3 and so on until 0.9) which span the entire probability range and implement the inverse FORM procedure for each of these η values. Therefore, we use the inverse FORM approach to quantify the uncertainty in k_E. The authors have explained the inverse FORM algorithm in detail in previous work (Sankararaman et al., 2013); in this section, the overall approach is briefly summarized and the algorithm is provided.

Both FORM and inverse FORM approaches linearize the curve represented by the equation $k_E = \mathtt{P}_{\boldsymbol{\lambda}}(\omega)$ and transform all the variables in $\boldsymbol{\Omega}$ to standard normal variables (Gaussian distribution with zero mean and unit variance) using well-

Algorithm 5 $\{k_E^{(i)}, \eta^{(i)}\}_{i=1}^N \leftarrow \mathtt{InverseFORM}(f_{\boldsymbol{\Omega}}(\omega), \mathtt{P}_{\boldsymbol{\lambda}})$

$N \leftarrow$ Number of η values to consider.
$M \leftarrow$ Number of elements in ω
{Example: $N = 9$, $\eta^{(i)} = 0.1 \times i$, $i = 1$ to N. }
for $i = 1$ **to** N (For every η value) **do**
  $\beta^{(i)} \leftarrow -\Phi^{-1}(\eta^{(i)})$
  $\omega_0 \leftarrow$ Select initial guess for optimization
  Convergence $= 0$
  $j = 0$ {Iteration number}
  while Convergence $\leftarrow 0$ **do**
    $\phi_j \leftarrow T(\omega_j)$ {Transformation to Std. Normal Space}
    $\phi_j \leftarrow [\phi_{jk}; k = 1 \text{ to } M]$
    $\alpha_j \leftarrow [\alpha_{jk}; k = 1 \text{ to } M]$ where $\alpha_{jk} = \frac{\partial \mathtt{P}_{\boldsymbol{\lambda}}}{\partial \phi_{jk}}$
    $\phi_{j+1} \leftarrow -\frac{\alpha_j}{|\alpha_j|} \times \beta^{(i)}$
    $\omega_{j+1} \leftarrow T^{-1}(\phi_{j+1})$ {Transformation to Original Space}
    if $\omega_{j+1} \approx \omega_j$ **then**
      Convergence $\leftarrow 1$
    end if
    $j \leftarrow j + 1$
  end while
  $k_E^{(i)} \leftarrow \mathtt{P}_{\boldsymbol{\lambda}}(\omega_j)$
end for

defined, popular transformation functions. Thus, k_E can be expressed as a linear sum of Gaussian variables, and therefore the probability distribution of k_E can be computed easily. The key point in these algorithms is that the point of linearization is chosen to be the Most Probable Point (MPP), i.e., the point of maximum likelihood value. For example, in a Gaussian distribution, the MPP is at the mean. Since each uncertain quantity in $\boldsymbol{\Omega}$ may have its own distribution, the MPP is computed in the standard normal space, where the origin has the highest likelihood of occurrence. However, the origin may not satisfy the equation $k_E = \mathtt{P}_{\boldsymbol{\lambda}}(\omega)$, and the point of linerization needs to lie on the curve represented by the equation $k_E = \mathtt{P}_{\boldsymbol{\lambda}}(\omega)$. Therefore, the problem reduces to estimating the minimum distance (measured from the origin, in the standard normal space) point on the curve represented by the equation $k_E = \mathtt{P}_{\boldsymbol{\lambda}}(\omega)$. This is posed as a constrained minimization problem, and solved using a well-known gradient-based optimization technique, as described in Algorithm 5. Once the minimum distance (denoted by β) has been evaluated, then it can be proved that $F_{K_E}(k_E) = \Phi(-\beta)$, where $\Phi(.)$ represents the standard normal distribution function. Algorithm 5 explains the numerical implementation of the inverse FORM approach.

In the above algorithm, note that the user needs to specify functions T and T^{-1} for transforming original space to standard normal space and from standard normal space to original space respectively. There are several types of transformation available in the literature (Haldar & Mahadevan, 2000) and any valid transformation may be used. Further, note that the gradient α needs to be calculated in the standard normal space. This depends on (*i*) the gradient in the original space; and (*ii*) the chosen transformation T.

Using the algorithm, the values of k_E corresponding to the

chosen values of η are first obtained, and then an interpolation technique can be used to obtain the entire CDF. Also, if the goal is to quickly obtain bounds on k_E, then we may consider two η values in either tail of the probability distribution (say, for example, η values of 0.1 and 0.9), and the corresponding probability bounds of k_E can be obtained.

4. CASE STUDY

As a case study, we perform battery prognostics on a planetary rover and present simulation-based results. The rover and its simulation are described in detail in (Balaban et al., 2011). The rover battery system consists of two parallel sets of 12 batteries in series to provide around 48 V. We are interested in predicting end-of-discharge (EOD), which is defined as the point when the voltage of a single battery drops below 2.5 V.

We consider two different scenarios for the rover. In both scenarios, the rover is provided a sequence of waypoints that must be visited. The rover travels through the waypoints in order until the batteries discharge. In the first scenario, the desired forward speed of the rover is the same when moving to each waypoint. In the second scenario, the desired forward speed is different depending on which waypoint is being approached. Since the needed power draw from the batteries depends on speed, the first scenario resembles a constant-amplitude loading situation, and the second scenario resembles a variable-amplitide loading scenario.

The battery prognostics architecture works as follows. The rover provides voltage measurements on all batteries, and the current going into the batteries. Because there are two sets of batteries in parallel, each battery sees only half the measured current. The measured current and voltage are fed into the battery prognoser. The battery prognoser uses an unscented Kalman filter (UKF) to perform state estimation (see (Julier & Uhlmann, 1997, 2004; Daigle, Saha, & Goebel, 2012) for details on the UKF). The state estimate is then fed into the predictor, which makes EOD predictions every 100 seconds.

In the remainder of the section, we describe the details of the underlying battery model used by the rover simulation and the battery prognoser, and provide simulation-based experimental results for different scenarios and comparing the different methods presented in Section 3.

4.1. Battery Model

The battery model is extended from that presented in (Daigle, Saxena, & Goebel, 2012). The model is based on an electrical circuit equivalent as shown in Fig. 2, following similar modeling approaches to (Chen & Rincon-Mora, 2006; Ceraolo, 2000). The large capacitance C_b holds the charge q_b of the battery. The nonlinear C_b captures the open-circuit potential and concentration overpotential. The R_{sp}-C_{sp} pair captures the major nonlinear voltage drop due to surface overpotential, R_s captures the so-called Ohmic drop, and R_p models the parasitic resistance that accounts for self-discharge. This empirical battery model is sufficient to capture the major dynamics of the battery while ignoring temperature effects and additional minor processes. The governing equations for the battery model are presented in continuous time below. The implementation of the proposed methodology considers a discrete-time version with a discrete time-step of 1 s.

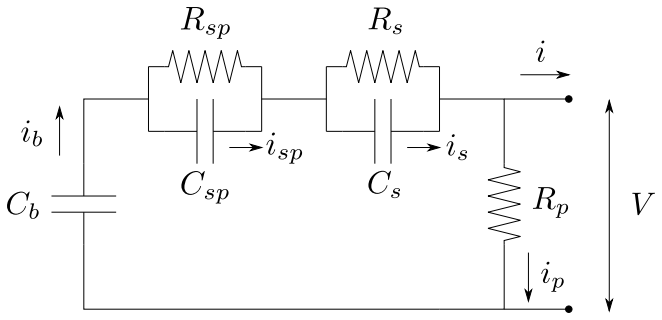


Figure 2. Battery equivalent circuit.

The state-of-charge, SOC, is computed as

$$SOC = 1 - \frac{q_{max} - q_b}{C_{max}}, \tag{5}$$

where q_b is the current charge in the battery (related to C_b), q_{max} is the maximum possible charge, and C_{max} is the maximum possible capacity. The resistance related to surface overpotential is a nonlinear function of SOC:

$$R_{sp} = R_{sp_0} + R_{sp_1} \exp\left(R_{sp_2}(1 - SOC)\right), \tag{6}$$

where R_{sp_0}, R_{sp_1}, and R_{sp_2} are empirical parameters. The resistance, and, hence, the voltage drop, increases exponentially as SOC decreases.

Voltage drops across the individual circuit elements are given by

$$V_b = \frac{q_b}{C_b}, \tag{7}$$

$$V_{sp} = \frac{q_{sp}}{C_{sp}}, \tag{8}$$

$$V_s = \frac{q_s}{C_s}, \tag{9}$$

$$V_p = V_b - V_{sp} - V_s, \tag{10}$$

where q_{sp} is the charge associated with the capacitance C_{sp}, and q_s is the charge associated with C_s. The voltage V_b is also the open-circuit voltage of the battery, which is a nonlinear function of SOC (Chen & Rincon-Mora, 2006). This is captured by expressing C_b as a third-order polynomial func-

Table 1. Battery Model Parameters

Parameter	Value
C_{b_0}	19.80 F
C_{b_1}	1745.00 F
C_{b_2}	−1.50 F
C_{b_3}	−200.20 F
R_s	0.0067 Ω
C_s	115.28 F
R_p	1×10^4 Ω
C_{sp}	316.69 F
R_{sp_0}	0.0272 Ω
R_{sp_1}	1.087×10^{-16} Ω
R_{sp_2}	34.64
q_{max}	3.11×10^4 C
C_{max}	30807 C

tion of SOC:

$$C_b = C_{b_0} + C_{b_1} SOC + C_{b_2} SOC^2 + C_{b_3} SOC^3 \quad (11)$$

The terminal voltage of the battery is

$$V = V_b - V_{sp} - V_s. \quad (12)$$

Currents associated with the individual circuit elements are given by

$$i_p = \frac{V_p}{R_p}, \quad (13)$$

$$i_b = i_p + i, \quad (14)$$

$$i_{sp} = i_b - \frac{V_{sp}}{R_{sp}}, \quad (15)$$

$$i_s = i_b - \frac{V_s}{R_s}. \quad (16)$$

The charges are then governed by

$$\dot{q}_b = -i_b, \quad (17)$$

$$\dot{q}_{sp} = i_{sp}, \quad (18)$$

$$\dot{q}_s = i_s. \quad (19)$$

In the case of the battery, the event E we are interested in predicting is EOD. T_E is specified as $V < V_{\text{EOD}}$, where $V_{\text{EOD}} = 2.5$ V.

The parameter values of the battery model are given in Table 1. All voltages are measured in Volts, resistances are measured in Ohms, charges are measured in Coulombs, and capacitances are measured in Coulombs per Volt (or Farads). Note that C_{b_0}, C_{b_1}, C_{b_2}, and C_{b_3} are simply fitting parameters in Eq. 11 and do not have physical meaning.

For the battery model, $\mathbf{x} = [q_b\ q_{sp}\ q_s]$, $\boldsymbol{\theta} = \varnothing$ (i.e., all parameters are assumed constant and no parameters will be estimated online), and $\mathbf{y} = [V]$. We consider power P to be the input to the battery, so $i = P/V$, i.e, $\mathbf{u} = [P]$. Here, we choose power as the input, rather than current as in previous battery prognostics works, because it is simpler to describe battery load in terms of power. For the same power demands from the rover onto the battery, current will increase as battery voltage decreases; it is necessary to capture this current-voltage relationship in order to use current as input. Therefore, it is much easier to predict future power usage than to predict future current draw, and hence, power is used as input.

4.2. Future Input Characterization

In the experiments presented in this section, we will consider only uncertainty in the state and in the future inputs (methods for dealing with process noise are described in (Daigle, Saxena, & Goebel, 2012; Sankararaman & Goebel, 2013)). Therefore we need to define $p(\boldsymbol{\lambda}_u)$ and the `constructU` function. We explore three methods that differ in complexity and the amount of system knowledge used.

The future input trajectory to a battery model depends on how the rover will be used. When moving from one waypoint to the next, the rover must turn towards the next waypoint while maintaining its foward speed (that is how the locomotion controller is designed to work). For the same forward speed, turning actually requires more power than going straight, because the rover must also move against the ground torques produced while turning in addition to the opposing forces produced when moving forwards. Further, because of the turning, the actual distance traveled between two waypoints is greater than the straight-line distance between them because the rover actually takes a curved path.

To correctly account for all these details, a system-level approach is required (Daigle, Bregon, & Roychoudhury, 2012). In this case, the whole rover and its locomotion controller would be considered as the system under prognosis. Thus, the whole rover would be simulated moving through the different waypoints, and this would define very precisely (depending on model fidelity) the power drawn from the batteries as a function of time. However, such an approach is more computationally expensive than considering only a single battery model.

A simpler approach is to assume that, in the current operation of the system, the future inputs to the system will look like the past inputs. That is, we can assume that the future power requirements will be constant with the mean and variance defined by the past power requirements over some time window. If the window size is large enough, then the differences in power that arise may be represented well enough in the statistics of the past behavior. Although simple, such an approach may work well in some circumstances.

As a middle ground, we can incorporate some system-level knowledge into predicting the future power requirements without resorting to a system-level simulation. We can do this by computing the mean and variance of the power draw and distance traveled between waypoint pairs for each forward

speed setting. Then, knowing the current rover location and the remaining waypoints, given a realization of each distance and power variable for the remaining waypoints and the desired forward speed when heading to each waypoint, we can compute the future power as a function of time based on the expected path the rover is going to take. This approach uses system knowledge, i.e., knowledge beyond the battery model, in order to compute useful predictions of the future inputs to the battery, and therefore makes useful EOD predictions for the battery, but without resorting to a system-level simulation.

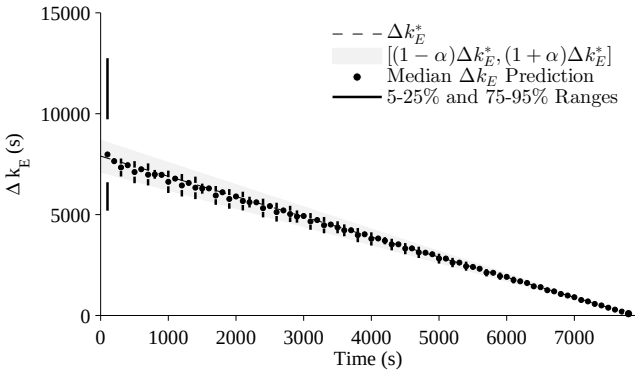
4.3. Results

In the simulation experiments considered in this section, all parameters are considered known exactly and no process noise is added. The two potential sources of uncertainty are related to the state estimate obtained by the UKF and the future input assumptions. Predictions are made every 100 s until EOD, and the accuracy and precision metrics are averaged over all these predictions. We use the relative accuracy (RA) metric as defined in (Saxena, Celaya, Saha, Saha, & Goebel, 2010) as a measure of accuracy and relative standard deviation (RSD) as a measure of spread. In the following plots, the * superscript indicates the ground truth values.

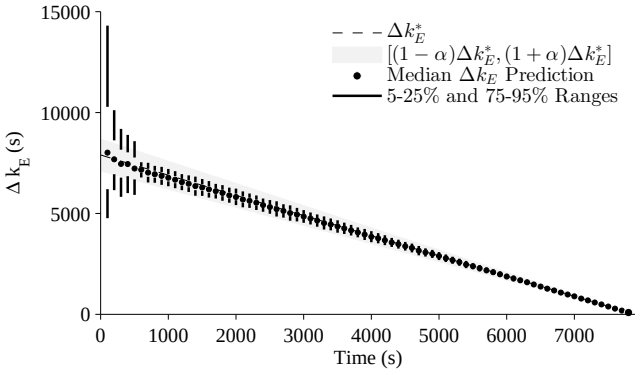
4.3.1. Constant-Loading Scenario

We consider first the scenario where the rover must move between equidistant waypoints at the same forward speed, resembling a constant-loading situation. Let us first assume that the future inputs (the battery power) are known exactly. There are 3 states in the battery model, so 7 sigma points are used by the UKF, and these are directly simulated forward to compute the k_E distribution using the sigma point weights. In this case, since uncertainty is limited only to that in the state estimate, predictions are both very accurate and precise, with RA averaging to 99.65% and RSD to 0.64%.

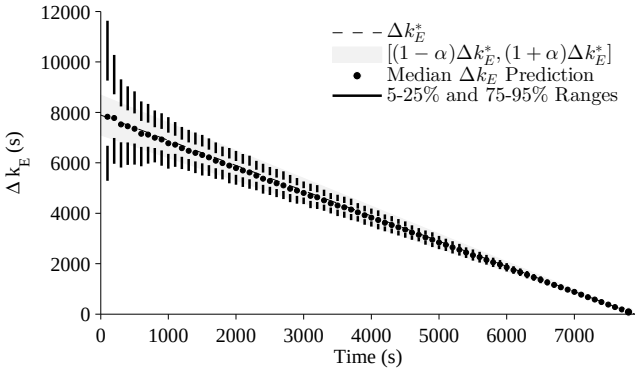
Now assume that the past power requirements are statistically representative of the future power requirements. Here, we consider Monte Carlo sampling with 3500 samples. We consider window sizes of 100 s, 500 s, and unlimited size. The results are shown in Fig. 3. In all three cases, the uncertainty starts initially very large, because the window is not large enough to accurately capture the statistics of the power usage. With a small window size (Fig. 3a), the statistics of the power usage averaged over the window fluctuate. The variance is larger when both turns and forward movements appear in the same window, and smaller when only forward movements are in the window. With a larger window size the variance will average to a larger value that accounts for both turns and forward movements, as seen in Figs. 3b and 3c. Since the past power usage turns out to be a good indicator of future power usage in this scenario, the results are fairly accurate, with RAs of 98.14%, 98.47%, and 97.53% for 100 s, 500 s, and unlimited window sizes, respectively. Corresponding RSDs are 3.83%, 5.57%, and 7.61%. The spread increases as window size increases since more variation is accounted for in larger windows.



(a) Window size of 100 s.



(b) Window size of 500 s.



(c) Unlimited window size.

Figure 3. Δk_E predictions using Monte Carlo sampling.

Using a window size of 500 s, the results using the UT are shown in Fig. 4. Here, the results are comparable to using Monte Carlo sampling with the same window size, with an average RA of 98.69% and RSD of 5.24%. The UT method, however, needs only 9 total samples, with there being only 3 states and one surrogate input variable to consider. This

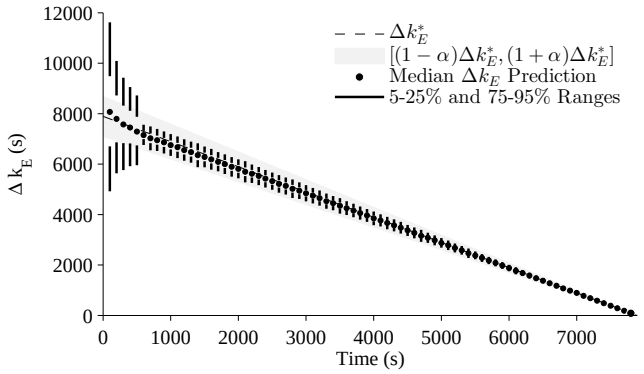


Figure 4. Δk_E predictions with a window size of 500 seconds using UT sampling.

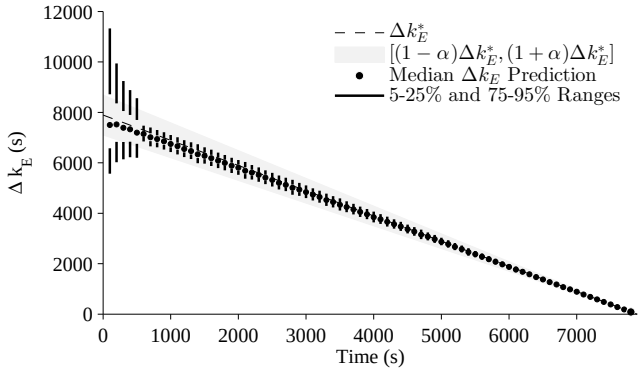


Figure 5. Δk_E predictions with a window size of 500 seconds using inverse FORM.

results in a substantial decrease in computational cost compared to the Monte Carlo approach. Fig. 5 shows the results using inverse FORM. The results are similar to using both Monte Carlo and UT sampling, and RA averages to 98.46% and RSD to 3.43%.

Using knowledge about the future waypoints to be visited, we can improve over using a window of past data to determine future inputs to the system. Fig. 6 shows the improved future input characterization method using Monte Carlo sampling with 3500 samples. The plots look the same for UT sampling and inverse FORM. The accuracy is comparable to the previous approach, with an average RA of 98.29% for Monte Carlo sampling, 98.32% for UT sampling, and 98.30% for inverse FORM. RSDs, however, are lower now since the future inputs are known with more precision than could be derived from a window of past samples. RSD averages to 1.61% for both Monte Carlo and UT sampling and 4.67% for inverse FORM.

4.3.2. Variable-Loading Scenario

We now consider the second scenario that uses the same waypoints as the previous scenario, but the rover is commanded

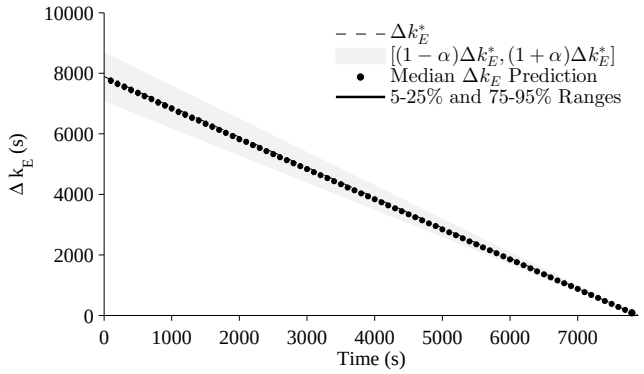
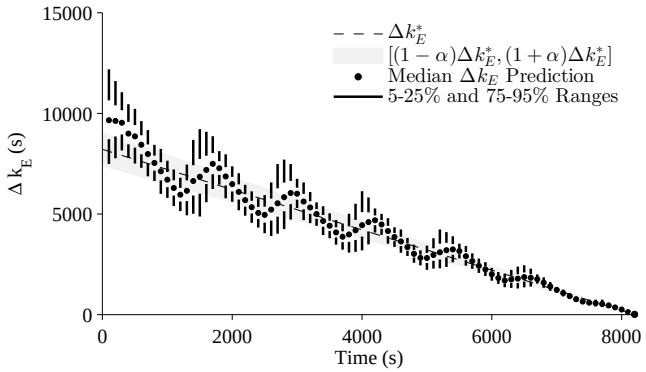


Figure 6. Δk_E predictions with improved future input characterization using Monte Carlo sampling.

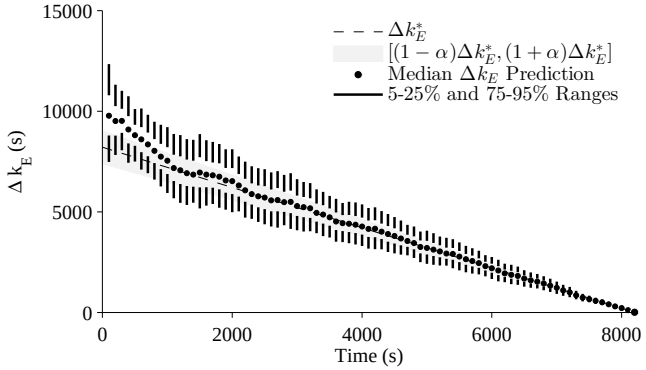
to go different speeds depending on which waypoint is being headed towards, resembling a variable-loading situation. Assuming the future inputs are known exactly, the average RA is 99.50% and the average RSD is 0.70%. The only uncertainty is in the state estimate.

Because the speed of the rover will change with each new waypoint, it is no longer correct to assume that past power requirements are representative of future power requirements. Fig. 7a shows the results when we incorrectly make this assumption for Monte Carlo sampling with 3500 samples, using a window size of 500 s. Clearly, the predictions are very inaccurate. RA averages to 90.53% and RSD to 13.74%. Using UT sampling we find similar results, with RA averaging to 90.12% and RSD to 13.54%. Using inverse FORM, RA averages to 90.59% and RSD to 9.24%. When the average speed of the rover in the future is greater than what is assumed based on the window of past samples, then Δk_E is overestimated. When the average speed is less than what is assumed, Δk_E is underestimated. Because the average speed over the window changes based on the previous waypoints within that window, the predictions fluctuate above and below ground truth. If the window size is increased, such that it accounts for all possible speed settings, then accuracy can be improved because the assumed average speed based on past samples will match the future average speed, however, spread will also increase since multiple speeds are considered in the window. Predictions with an unlimited window size are shown in Fig. 7b. The predictions initially fluctuate as the window begins to fill up, but by 2000 s the predictions have smoothed out. The spread is clearly larger than with the smaller window size, but predictions are more accurate once the window contains all potential future speeds. In this case, RA improves to 96.62% but RSD increases to 17.47%.

Predictions can be improved by using system knowledge to help characterize the future inputs. In this case, the future power as a function of time is computed based on the rover's current location, the remaining waypoints, and the desired



(a) Window size of 500 s.



(b) Unlimited window size.

Figure 7. Δk_E predictions using Monte Carlo sampling.

speeds when heading to each waypoint, and measured statistics on average power between waypoints and average distance traveled (to account for turns). Fig. 8 shows the results using Monte Carlo sampling. The plots for UT sampling and inverse FORM look the same. All three methods are now clearly very accurate and precise. RA averages to 98.85% and RSD to 1.95% for Monte Carlo sampling, 98.82% and 1.91% for UT sampling, and 97.77% and 2.72% for inverse FORM. Unlike when using a window, here knowledge of the future waypoints and desired speeds allows accurate predictions to be made from the start of the scenario, and with very little spread. That is, the future inputs are well-known and so predictions are very close to the optimal.

In the case above, for UT sampling, there are 3 states to consider and at most 100 surrogate input variables. For the input variables, there are two variables associated with each remaining waypoint, one for the power that will be consumed heading towards the waypoint and one for the distance to travel to a waypoint from the previous waypoint (due to turning while moving forward, the distance is more than the linear distance between the waypoints). Since there are 50 waypoints, there are 100 random variables needed. This yields $2(103) + 1 = 207$ samples, which is relatively small compared to what Monte Carlo sampling would require to achieve

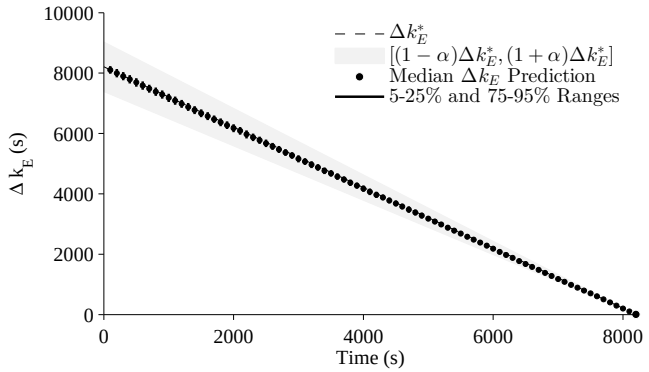


Figure 8. Δk_E predictions with improved future input characterization using Monte Carlo sampling.

the same performance. As the rover visits waypoints, the number is reduced, so 207 samples is only the maximum. When only a single waypoint is left, only $2(5) + 1 = 11$ sigma points are needed.

4.4. Discussion

The two scenarios demonstrate the importance of the future input characterization problem. Even though the rover is a complex system, in the first scenario, the simple assumption that the future inputs will look like the past inputs was sufficient for accurate and precise predictions. The additional power required by turns was captured using the statistics of a window of past samples, so that they did not have to be explicitly accounted for and assuming constant future inputs was sufficient. Using system-level information about the waypoints the rover would visit improved significantly on the uncertainty associated with the future inputs but did not significantly impact accuracy.

In the second scenario, the assumption that the future inputs look like the past inputs did not provide as accurate or precise results as with the first scenario. Performance could have been potentially worse if the rover did not cycle through different speed settings and instead, for example, always increased the speed for the next waypoint. In this scenario, truly accurate predictions could be made only when system-level knowledge was utilized to predict the battery inputs. This approach still made some simplifying assumptions allowing a component-level prognostics approach to still be used.

Given a particular method for future input characterization, Monte Carlo sampling, UT sampling, and inverse FORM all had comparable accuracy and precision. With Monte Carlo sampling, a large number of samples were used and with smaller numbers of samples, performance decreases, so the number of samples required depends on the prognostics performance requirements. In this sense Monte Carlo sampling has an advantage because its computational complexity can be tuned. In addition it is the relatively simplest approach

to implement of the three methods. A disadvantage is that it is a stochastic algorithm, which can be problematic for verification and certification procedures (Daigle, Saxena, & Goebel, 2012). UT sampling, in contrast, is a determinstic algorithm, and it selects only the minimal number of samples and this number grows only linearly with the number of random variables. A disavdvantage though is that it computes only mean and variance of the predictions, which may not be enough information in some cases. Inverse FORM, on the other hand, is not only deterministic but also allows control of both computational complexity and accuracy by selecting desired CDF values and computing the corresponding percentile values of k_E. The probability distribution of k_E can be reconstructed from that information and any desired statistical moments may be calculated. For each inverse CDF calculation, three to four iterations are usually required for optimization convergence. If the number of random variables is denoted by n (length of vector ω), each iteration of Inverse-FORM requires $n+1$ sample evaluations of $\mathtt{G}$, where one evaluation is required for computing k_E and n evaluations for computing the gradient vector of k_E. Therefore, if it is desired to repeat inverse FORM for k different CDF values, then $k \times 4 \times (n+1)$ evaluations of $\mathtt{P}$ are required. Thus, the computational complexity linearly increases with the number of random variables and results in increased information regarding uncertainty.

5. Conclusions

In this paper, we provided a general formulation of the prognostics problem and its uncertainty. Given descriptions of the sources of uncertainty, i.e., state uncertainty, parameter uncertainty, future input uncertainty, and process noise, we provided an algorithmic framework for incorporating this uncertainty into the predictions. With the novel concept of surrogate variables, we presented three methods for propagating the uncertainty: Monte Carlo sampling, unscented transform sampling, and the inverse first-order reliability method. Using battery prognostics on a planetary rover as a case study, we proposed two future input characterization methods and compared the performance of the different prediction algorithms for these methods for different scenarios in simulation. All approaches had similar performance, yet each offer different advantages and disadvantages that suggest when one would be preferred over another.

In future work, we will further investigate these ideas on other systems, and further develop the uncertainty quantification framework. While the proposed methods are promising for estimating uncertainty in prognostics, their applicability to multi-modal probability distributions, particular in the context of remaining useful life estimation, needs to be investigated. Further, we will also focus on model uncertainty quantification and develop methods for estimating model errors and model parameter uncertainty separately, instead of simply using lumped process noise terms.

Acknowledgment

This work was funded in part by the NASA System-wide Satefy Assurance Technologies (SSAT) project under the Aviation Safety (AvSafe) Program of the Aeronautics Research Mission Directorate (ARMD), and by the NASA Automated Cryogenic Loading Operations (ACLO) project under the Office of the Chief Technologist (OCT) of Advanced Exploration Systems (AES).

References

Arulampalam, M. S., Maskell, S., Gordon, N., & Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, *50*(2), 174–188.

Balaban, E., Narasimhan, S., Daigle, M., Celaya, J., Roychoudhury, I., Saha, B., ... Goebel, K. (2011, September). A mobile robot testbed for prognostics-enabled autonomous decision making. In *Annual conference of the prognostics and health management society* (p. 15-30). Montreal, Canada.

Ceraolo, M. (2000, November). New dynamical models of lead-acid batteries. *IEEE Transactions on Power Systems*, *15*(4), 1184–1190.

Chen, M., & Rincon-Mora, G. A. (2006, June). Accurate electrical battery model capable of predicting runtime and I-V performance. *IEEE Transactions on Energy Conversion*, *21*(2), 504 - 511.

Daigle, M., Bregon, A., & Roychoudhury, I. (2012, September). A distributed approach to system-level prognostics. In *Annual conference of the prognostics and health management society* (p. 71-82).

Daigle, M., & Goebel, K. (2010, October). Improving computational efficiency of prediction in model-based prognostics using the unscented transform. In *Proc. of the annual conference of the prognostics and health management society 2010.*

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

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

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

Haldar, A., & Mahadevan, S. (2000). *Probability, reliabil-*

ity, and statistical methods in engineering design. John Wiley & Sons, Inc.

Julier, S. J. (1998, April). A skewed approach to filtering. In *Proc. aerosense: 12th int. symp. aerospace/defense sensing, simulation and controls* (Vol. 3373, p. 54-65).

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

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

Ling, Y., Shantz, C., Mahadevan, S., & Sankararaman, S. (2011). Stochastic prediction of fatigue loading using real-time monitoring data. *International Journal of Fatigue*, *33*(7), 868–879.

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

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

Saha, B., Quach, C. C., & Goebel, K. (2012, March). Optimizing battery life for electric UAVs using a Bayesian framework. In *Proceedings of the 2012 IEEE Aerospace Conference.*

Sankararaman, S., Daigle, M., Saxena, A., & Goebel, K. (2013, March). Analytical algorithms to quantify the uncertainty in remaining useful life prediction. In *Proceedings of the 2013 IEEE Aerospace Conference.*

Sankararaman, S., & Goebel, K. (2013, April). Uncertainty quantification in remaining useful life of aerospace components using state space models and inverse form. In *Proceedings of the 15th Non-Deterministic Approaches Conference.*

Sankararaman, S., Ling, Y., Shantz, C., & Mahadevan, S. (2011). Uncertainty quantification in fatigue crack growth prognosis. *International Journal of Prognostics and Health Management*, *2*(1).

Saxena, A., Celaya, J., Saha, B., Saha, S., & Goebel, K. (2010). Metrics for offline evaluation of prognostic performance. *International Journal of Prognostics and Health Management*, *1*(1).

BIOGRAPHIES

Matthew Daigle received the B.S. degree in Computer Science and Computer and Systems Engineering from Rensselaer Polytechnic Institute, Troy, NY, in 2004, and the M.S. and Ph.D. degrees in Computer Science from Vanderbilt University, Nashville, TN, in 2006 and 2008, respectively. From September 2004 to May 2008, 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. During the summers of 2006 and 2007, he was an intern with Mission Critical Technologies, Inc., at NASA Ames Research Center. From June 2008 to December 2011, he was an Associate Scientist with the University of California, Santa Cruz, at NASA Ames Research Center. Since January 2012, he has been with NASA Ames Research Center as a Research Computer Scientist. His current research interests include physics-based modeling, model-based diagnosis and prognosis, simulation, and hybrid systems. Dr. Daigle is a member of the Prognostics and Health Management Society and the IEEE.

Shankar Sankararaman received his B.S. degree in Civil Engineering from the Indian Institute of Technology, Madras in India in 2007 and later, obtained his Ph.D. in Civil Engineering from Vanderbilt University, Nashville, Tennessee, U.S.A. in 2012. His research focuses on the various aspects of uncertainty quantification, integration, and management in different types aerospace, mechanical, and civil engineering systems. His research interests include probabilistic methods, risk and reliability analysis, Bayesian networks, system health monitoring, diagnosis and prognosis, decision-making under uncertainty, treatment of epistemic uncertainty, and multidisciplinary analysis. He is a member of the Non-Deterministic Approaches (NDA) technical committee at the American Institute of Aeronautics, the Probabilistic Methods Technical Committee (PMC) at the American Society of Civil Engineers (ASCE), and the Prognostics and Health Management (PHM) Society. Currently, Shankar is a researcher at NASA Ames Research Center, Moffett Field, CA, where he develops algorithms for uncertainty assessment and management in the context of system health monitoring, prognostics, and decision-making.