Particle Filtering for Diagnosis and Prognosis of Anomalies in Rocket Engine Tests

Bikramjit Banerjee * and Landon Kraemer †

The University of Southern Mississippi, 118 College Dr. #5106, Hattiesburg, MS 39406, USA

Wanda Solano ‡

NASA/NCCIPS, 9313 Cypress Loop Road, Stennis Space Center, MS 39529, USA

We present a novel technique for anomaly detection and prognosis in sensor data from rocket engine test stands. We apply a combination of particle filtering and machine learning approaches to capture the model of nominal operations, and use voting techniques in conjunction with particle filtering to detect anomalies in test runs. We use two approaches – pure particle filtering and pure machine learning – for prognosis. Our experiments on test stand sensor data show successful detection of a known anomaly in the test data, while producing almost no false positives. Both prognostic approaches, however, predict no further impact had the test been completed, perhaps indicating that the anomaly was innocuous.

I. Introduction & Objectives

We present the application of two well-known AI techniques – Bayesian filtering and machine learning – to the problems of diagnosis and prognosis of anomalies in sensor network data. Our objective was to develop a system that post-processes a csv file showing the sensor readings and activities (time-series) from a rocket engine test, and detect any anomalies that might have occurred during the test, as well as predict the future evolution of these (and other) anomalies if the test was allowed to continue. The output was required to be in the form of the names of the sensors that show anomalous behavior, and the start and end time of each anomaly, both diagnosed and predicted. Since our approach was model-based, we needed to automatically learn a model of nominal behavior from tests that were marked nominal. In this paper, we will describe this system and show experimental results that demonstrate that it has successfully detected a known anomaly in a given test stand data set, and delivered a prognosis that matches the broad conclusion of the test engineers.

The paper is organized as follows. In section III we present the theoretical background, viz., dynamic Bayesian networks and the particle filtering framework that underlies our approach. In section IV we present our anomaly detection model and explain how it is tied to the particle filtering framework. In section V we describe anomaly detection module in detail, and present the experimental results in section V.D. In section VI we present the prognosis module and mention the experimental results. We present our conclusions and future work in section VII.

II. Motivation

The genesis of our ideas on fast anomaly detection and prognosis in sensor data, lies in the several limitations of existing approaches. The prevailing approach to anomaly detection in spacecraft sensor data has been to employ many human experts, aided by simple range-checks that signal when a particular variable goes out of a pre-determined range.[1] Given the very large number of variables involved, and the large

*Assistant Professor, School of Computing

†Graduate Research Assistant, School of Computing, AIAA Student Member

‡Principal Technologist, National Center for Critical Information Processing and Storage (NCCIPS)

disparity between data sampling frequency and human reaction times, it is a tedious and an error-prone process. In the recent past, several model-based, automatic anomaly detection systems were developed by encoding knowledge from human experts, such as Livingstone, TEAM-RT, RODON, SHINE and others. However, engineering these systems have also been labor-intensive, as well as incomplete.

In order to significantly reduce the involvement of domain experts, several data-driven approaches have been proposed, where models are automatically acquired from the data, thus bypassing the cost and effort of building system models. Many supervised learning methods are known (such as neural networks and support vector machines) that can efficiently learn operational and fault models, given large amounts of both nominal and fault data. However, for domains such as SSME and RETS data, the amount of anomalous data that is actually available is relatively small, making most supervised learning methods rather ineffective (an important exception is one-class support vector machines, although it can be less robust[2]). Despite this, Guo and Musgrave[3] applied neural networks to sensor validation for the SSME with some success, and follow-up work demonstrated a greater efficacy for support vector machines. In general, however, supervised methods have met with limited success in anomaly detection. We also apply neural networks, but in an innovative way so that they only need nominal data. Thus we overcome the limitation inherent in supervised learning.

Recently, model-free techniques have gained momentum. Schwabacher[1] has reported the results of applying two unsupervised anomaly detection algorithms, Orca (nearest neighbor classifier with scale-up) and GritBot (subset based outlier detection), to data from rocket propulsion testbeds (SSME and E-1 test stand at Stennis Space Center). Four different anomalies were detected, including a brief de-correlation between two strongly correlated redline enablers. However, experts did not feel any of these could be considered anomalies.

One of the most popular unsupervised learning technique is *clustering*.[4] Iverson's Inductive Monitoring System (IMS)[5] applies the clustering approach to divide the nominal training data into clusters, representing different modes of the system. When new data fails to fit into any of the known clusters, it is flagged as an anomaly, with its distance from the nearest cluster indicating the "strength" of the anomaly. Retrospective analysis has shown that this method might have succeeded in identifying an anomaly in the temperature sensors in the left wing of space shuttle Columbia, well before the disaster. However, these systems have turned out to be hyper-sensitive in anomaly detection, and incapable of prognosis.

Another application of the clustering approach, in conjunction with entropy measures, was performed by Agogino and Tumer[6] on SSME data. Their approach was two-pronged: on the one hand they performed entropy analysis over the entire set of sensors to detect anomalies that have broad system-wide impact. On the other hand, they also clustered the sensors themselves (as opposed to sensor data) to isolate their impacts that have a more localized nature. They used *cluster entropy* to detect anomalies that are specific to certain subsystems, and whose impact is spread over a small subset of correlated sensors.

Despite the attempt by Agogino and Tumer to exploit the local and global correlations in sensors, the fundamental problem with all of these unsupervised and supervised approaches is that they assume that the data are *i.i.d*, i.e., independent and identically distributed, which is violated in typical RETS data. None of these techniques naturally exploit the temporal information inherent in time series data from the sensor networks. There are correlations among the sensor readings, not only at the same time, but also *across time*. But hardly any of these approaches have explicitly identified and exploited such correlations.

Given these limitations of model-free methods, there has been renewed interest in model-based methods, specifically graphical methods that explicitly reason temporally. Martin[2,7] acknowledges the i.i.d. assumption problem with previous approaches, and proposes a Gaussian Mixture Model (GMM) in a Linear Dynamic System, i.e., a linear Markov chain of hidden states, $\boldsymbol{x}_k$ (unfilled circles), and observations, $\boldsymbol{z}_k$ (filled circles), as shown in Figure 1. In this figure, the arrows show the causal dependence, so the horizontal arrows imply temporal dependence (albeit Markovian, i.e., each state only depends on the previous state). This work assumes that the multi-dimensional SSME data is a mixture of multi-variate Gaussians, and fits a given number of Gaussian clusters with the help of the well-known Expectation Maximization (EM) algorithm. The parameters thus learned, are used for calculating the joint distribution of the observations ($\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots$ in Figure 1). However, it should be noted that the GMM assumption is essentially an approximation, and Martin himself acknowledges[7] the potential viability of non-parametric density estimators. This is precisely the key idea underlying our approach.

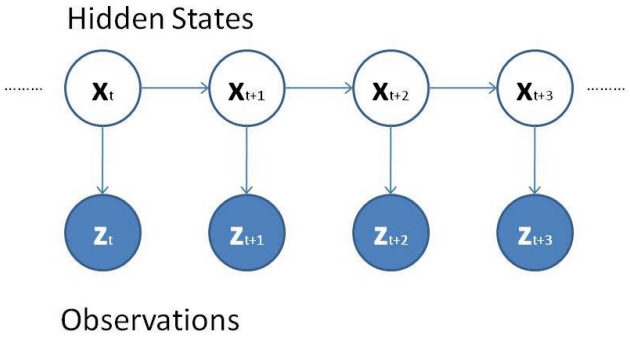


Figure 1. Linear Dynamic System

III. Technical Background

The conceptual model behind our approach is that of a dynamic Bayesian network; in particular a linear Markov chain of hidden states, $\boldsymbol{x}_k$ (unfilled circles), and observations, $\boldsymbol{z}_k$ (filled circles), as shown in Figure 1. Conceptually, the hidden state $\boldsymbol{x}_k$ of the system changes due to some activities (test commands) in a way that can be learned from nominal tests. We can track such changes due to the observations $\boldsymbol{z}_k$ (sensor readings) which are tied to the hidden states. For this task of tracking, we have employed a technique called *particle filtering* which we describe next.

III.A. Particle Filtering

Filtering is the problem of estimating the current state, $\boldsymbol{x}_t$, given the observations until now, $\boldsymbol{z}_{1:t}$. Typically the states are hidden, with access limited to merely the series of observations. So we need to construct the pdf $p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t})$, assuming that the prior $p(\boldsymbol{x}_0|\boldsymbol{z}_0) = p(\boldsymbol{x}_0)$ is known, or is uniform ($\boldsymbol{z}_0$ is the null observation). In principle, this pdf can be obtained recursively in two stages: prediction and update, as discussed below.

III.A.1. Prediction

Suppose the posterior at the previous time instant $(t-1)$ was available, i.e., $p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{1:t-1})$. This could then be used to generate a prediction regarding $\boldsymbol{x}_t$, with a transition density, $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$, using the Chapman Kolmogorov equation

$$p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t-1}) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{1:t-1})d\boldsymbol{x}_{t-1}$$

Note that the transition density assumes that the underlying (hidden) states undergo transition (either passively, e.g., a Markov process, or under some actions from an agent) that is conditionally independent of the observations, given the previous state. Formally, $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{z}_{1:t-1})$. Furthermore, this transition may follow a noisy process; all we need is to be able to gather statistics on the noise, e.g., mean and variance, either from measurement or from prior knowledge of the nature of the process. It is common to assume that this noise is white.

III.A.2. Update

Given the prediction of the likelihood of $\boldsymbol{x}_t$, based on all prior observations (i.e., $\boldsymbol{z}_{1:t-1}$), the current observation ($\boldsymbol{z}_t$) plays a critical role: It can be utilized to validate the prediction, and thus update the prior. By Bayes' rule

$$p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t}) = \frac{p(\boldsymbol{z}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t-1})}{\int p(\boldsymbol{z}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t-1})d\boldsymbol{x}_t} \tag{1}$$

where $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ is the observation density, usually defined by a measurement model, and associated noise. In other words, the update stage uses the new observation $\boldsymbol{z}_t$ to refine the prior density to obtain the posterior density of the current state.

III.B. Sequential Importance Sampling

The recurrence relations defined above form the basis of the optimal Bayes solution. Unfortunately, the solution can seldom be generated analytically. Solution methods such as Kalman filtering and extended Kalman filtering crucially depend on approximating the densities as Gaussians. Another approach is to use sequential Monte Carlo filtering, where the posterior density ($p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t})$; equation 1), instead of being approximated, is represented by a set of random samples or particles (with weights given by the posterior), and the estimates are generated on the basis of these samples. This sequential Monte Carlo approach is one way of implementing the particle filtering method.

Let's say the ith particle ($i = 1 \ldots N$) represents the hypothesis $\boldsymbol{x}_{t-1}^i$ at time step $t-1$. The particle filtering algorithm replaces this sample at time t by sampling from the distribution $p(\boldsymbol{x}_t^i|\boldsymbol{x}_{t-1}^i)$, and then updating its weight as $w_t^i = p(\boldsymbol{z}_t|\boldsymbol{x}_t^i)$. After all the N particles have been updated this way, a new set of N particles is sampled with replacement from the weight distribution, and the entire process is repeated.

IV. Particle Filtering for Anomaly Detection

Our key insight is that the particle filtering technique for localization can be naturally applied to anomaly detection from sensor readings, with significant benefits compared to existing approaches. In this case, the mapping to the anomaly detection problem is given by:

- the (hidden) state variable, $\boldsymbol{x}_t$, is the unknown nominal correlation model among the sensor readings (both spatially and temporally), that the algorithm "localizes" over a few initial iterations, and thereafter tracks this nominal operation.
- the (noisy) sensor readings themselves constitute the observation vector, $\boldsymbol{z}_t$.

Our correlation model is simply a matrix, M, of size $n \times n$ where n is the number of sensors. This is also the hidden state:

$$\boldsymbol{x} = M.$$

Given the sensor observations at time t, $\boldsymbol{z}_t$, we predict that the sensor observations at $t+1$ would be

$$\boldsymbol{z}_{t+1} = M_t.\boldsymbol{z}_t, \ i.e., \boldsymbol{z}_{t+1} = \boldsymbol{x}_t.\boldsymbol{z}_t \tag{2}$$

This generates the observation density $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$. Since the matrix $M_t = \boldsymbol{x}_t$ is unknown, this is the variable we track by particle filtering. The only other unknown is the transition density $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, a_t)$, which in this case depends not only on the previous state but also on the activity a_t (such as a command) performed at that time during the test. For this, we train a neural network based on the nominal tests.

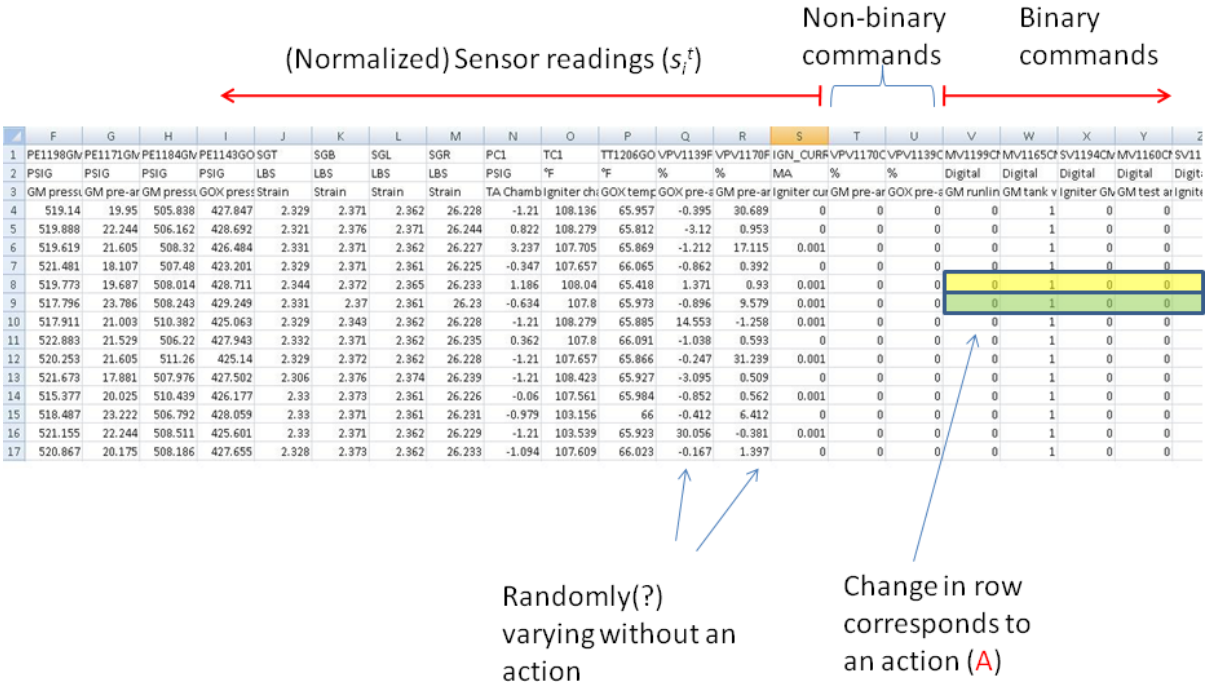
V. System Description

Figure 2 shows a typical input test data file. Our system automatically identifies the binary columns and interprets them as the commands. It identifies an activity a_t as the *change* in some binary column between two successive rows. All non-binary columns are interpreted as sensors.

V.A. Pre-processing

The sensor columns are first normalized individually, so that each reading is in the range $[-1, 1]$ where 1 corresponds to the largest magnitude of the values for that particular sensor. Normalizing the value in this way allows us to relate sensors with relatively different value ranges. According to equation 2, the ith row of our $n \times n$ model, $\boldsymbol{x}_t = M_t$, represents a linear combination of all of the sensor values, $\boldsymbol{z}_t$, to produce the predicted value of sensor i at time $t+1$, i.e., z_t^i. Since each of these linear combinations is used to predict the value of exacly one sensor and we know that the value of a sensor must lie in the range $[-1, 1]$, we reduce our search space by restricting each row to an affine combination.

The number of frames (rows in the input csv file) that we have to consider directly affects the runtime during both training and testing. Datasets that have data captured at a high frequency will have many frames, but if there is relatively little change from frame to frame, we can reduce the amount of frames without much loss of information. So, in our preprocessing stage we average the frames together at some



	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	PE1198GM	PE1171GM	PE1184GM	PE1143GO	SGT	SGB	SGL	SGR	PC1	TC1	TT1206GO	VPV1139F	VPV1170F	IGN_CURR	VPV1170C	VPV1139C	MV1199CI	MV1165CI	SV1194CM	MV1160CI	SV11
2	PSIG	PSIG	PSIG	PSIG	LBS	LBS	LBS	LBS	PSIG	°F	°F	%	%	MA	%	%	Digital	Digital	Digital	Digital	Digit
3	GM pressu	GM pre-ar	GM pressu	GOX press	Strain	Strain	Strain	Strain	TA Chamb	Igniter ch	GOX temp	GOX pre-s	GM pre-ar	Igniter cur	GM pre-ar	GOX pre-s	GM runlin	GM tank v	Igniter GN	GM test ar	Ignite
4	519.14	19.95	505.838	427.847	2.329	2.371	2.362	26.228	-1.21	108.136	65.957	-0.395	30.689	0	0	0	0	1	0	0	
5	519.888	22.244	506.162	428.692	2.321	2.376	2.371	26.244	0.822	108.279	65.812	-3.12	0.953	0	0	0	0	1	0	0	
6	519.619	21.605	508.32	426.484	2.331	2.371	2.362	26.227	3.237	107.705	65.869	-1.212	17.115	0.001	0	0	0	1	0	0	
7	521.481	18.107	507.48	423.201	2.329	2.371	2.361	26.225	-0.347	107.657	66.065	-0.862	0.392	0	0	0	0	1	0	0	
8	519.773	19.687	508.014	428.711	2.344	2.372	2.365	26.233	1.186	108.04	65.418	1.371	0.93	0.001	0	0	0	1	0	0	
9	517.796	23.786	508.243	429.249	2.331	2.37	2.361	26.23	-0.634	107.8	65.973	-0.896	9.579	0.001	0	0	0	1	0	0	
10	517.911	21.003	510.382	425.063	2.329	2.343	2.362	26.228	-1.21	108.279	65.885	14.553	-1.258	0.001	0	0	0	1	0	0	
11	522.883	21.529	506.22	427.943	2.332	2.371	2.362	26.235	0.362	107.8	66.091	-1.038	0.593	0	0	0	0	1	0	0	
12	520.253	21.605	511.26	425.14	2.329	2.372	2.362	26.228	-1.21	107.657	65.866	-0.247	31.239	0.001	0	0	0	1	0	0	
13	521.673	17.881	507.976	427.502	2.306	2.376	2.374	26.239	-1.21	108.423	65.927	-3.095	0.509	0	0	0	0	1	0	0	
14	515.377	20.025	510.439	426.177	2.33	2.373	2.361	26.226	-0.06	107.561	65.984	-0.852	0.562	0.001	0	0	0	1	0	0	
15	518.487	23.222	506.792	428.059	2.33	2.371	2.361	26.231	-0.979	103.156	66	-0.412	6.412	0	0	0	0	1	0	0	
16	521.155	22.244	508.511	425.601	2.33	2.371	2.362	26.229	-1.21	103.539	65.923	30.056	-0.381	0.001	0	0	0	1	0	0	
17	520.867	20.175	508.186	427.655	2.328	2.373	2.362	26.233	-1.094	107.609	66.023	-0.167	1.397	0	0	0	0	1	0	0	

Figure 2. Data file (csv) and interpretation.

specified interval, and thus scale down the number of frames unless there are too few frames between two actions (a_t and a_{t+1}) to allow such aggregation.

V.A.1. Sensor Selection

The model M_t used in equation 2 is of size $O(n^2)$, where n is the number of sensors used. Since we use a neural network to learn how the model changes (to generate $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, a_t)$) when an action occurs (see section V.B), and the complexity of training (and even applying) the neural network is $O(n^4)$, minimizing the number of sensors can greatly reduce the runtime of our system. There is a tradeoff in this selection problem. Omitting a sensor improves the runtime but reduces the amount of information we have about the system, and if certain groups of sensors are omitted, it may be impossible to detect some anomalies. Then, ideally, we would like the smallest set of sensors that best represents the state of the system. This could be difficult to do manually, so we have created a principled way to choose this set of sensors.

In order to obtain this set, we first partition the sensors into groups such that the sensors within each group are highly correlated with each other, but those in different groups are poorly correlated. Although this could be accomplished by traditional clustering algorithms, we did not want to impose limits on the number or the sizes of the clusters. We developed an efficient greedy algorithm for this purpose, which is shown as Algorithm 1.

From each of these clusters with size greater than one, we select the sensor that is most correlated with the other sensors in that cluster (i.e., most representative of this cluster) and add it to our set of selected sensors. After selecting such sensors (resulting in say k sensors), we sort the ungrouped sensors - i.e. groups of size one - in descending order based upon how uncorrelated they are to all of the other sensors. Then, we select the top $n - k$ of these sensors, where n depends on how many more sensors we have time to accommodate. This way we select n sensors.

Note that a major consequence of this method of sensor selection is that a non-systemic problem, i.e. failure of an individual sensor that affects no other sensor (even the ones it is supposed to be correlated with), cannot be detected unless that sensor is included. However, if an omitted sensor is known to have exhibited an anomaly, and no anomaly is observed using our method, it is likely that the anomaly is not due to a systemic problem but local to that sensor, and can most likely be overlooked.

Algorithm 1 Greedy Correlation Clustering

{Input $g(\{a\},\{b\})$, the correlation values for all sensors $a, b \in A$. Returns a partition of A}
(Initialize) $CS \leftarrow \{\{a_1\},\{a_2\},\ldots,\{a_n\}\}$, and best-gain > 0.
while best-gain > 0 **do**
  for each pair of sensor-clusters $C, C' \in CS$ **do**
    $CS' \leftarrow ((CS \setminus C) \setminus C') \cup (C \cup C')$
    $h(C, C') \leftarrow g(C, C') + \max(0, \max_{C'', C''' \in CS'} g(C'', C'''))$
  end for
  $(P, Q) \leftarrow \arg\max_{C, C' \in CS} h(C, C')$
  best-gain $\leftarrow h(P, Q)$
  if best-gain > 0 **then**
    $C_m \leftarrow P \cup Q$
    $CS \leftarrow ((CS \setminus P) \setminus Q) \cup C_m$
    for $C \in (CS \setminus C_m)$ **do**
      $g(C_m, C) \leftarrow g(P, C) + g(Q, C)$
    end for
  end if
end while
Return CS

V.A.2. Action Selection

The size of the action set (the binary columns in the data file, see Figure 2) affects the size of the input to the neural network (see section V.B). While it is not the most prominent factor of the complexity of using the neural network, it still has appreciable effect on the runtime. Thus, we would like to choose the minimal set of actions that represents the state changes in the system. This requires domain knowledge, and *we have not automated this process.*

Even though the action set has been minimized by manual selection, we still may have multiple actions that occur when a state change occurs. Since the actions can occur within a few frames of each other this can be problematic. Since there are few frames between the actions, we will not be able to properly localize, and this could cause us to learn bad transitions. Also, since we train the neural network at each transition, this will increase the number of times we have to train the neural network. We solve this problem by simply merging actions together if some amount of time has not passed between them.

V.A.3. Variance Masks

We also offer *variance masks* as a screening method for sensors. A variance mask is a binary vector with an element corresponding to each sensor in the sensor set. If a sensor is masked (zero), then both the row and the column corresponding to that sensor in the model matrix M_t are zeroed out, and that sensor is excluded from the error function. Essentially, while a sensor is masked by the error function it does not exist. We offer two types of variance masking, *inactivity masking* and *overactivity masking.* Inactivity masking considers each sensor's variance over the entire data set. If the variance does not exceed some threshold, the sensor is considered inactive and will be masked out over the entire dataset. The purpose of this type of masking is to remove sensors that maintain a near-constant value the entire run, as if these sensors were idling throughout the run. If we were to leave such a sensor in, we could potentially learn a model where a significant portion of the prediction for another sensor is based on it. Ideally, no useful sensors should exhibit this behavior. If one is found that behaves this way it can actually be removed from the sensor set entirely.

Overactivity masking considers a sensor's variance over a segment (a_t to a_{t+1}, not the entire dataset). If that sensor's variance exceeds some threshold, the sensor is masked for that segment only. The idea behind this is that there may be segments for which a sensor's value may vary wildly even in the nominal case, and ignoring the sensor for that segment could reduce false positives. Caution should be taken when using this type of masking because if we have, for instance, an orderly linear change over a segment, the variance may be high due to the change in magnitude.

V.B. Training the Transition Model

Our primary assumption is that the underlying model $\mathcal{M}$ (which is tracked by hypothesis samples M_t) will only drastically change when actions occur, and if the current model does not explain the observations well enough beyond actions, we say that an anomaly has occurred. This requires us to learn a nominal model, since without knowing what is "normal", we cannot decide what is "abnormal". To this end, we first learn how to transition models (i.e., $(M_t, a_t) \rightarrow M_{t+1}$) from action to action (i.e., between a_t and a_{t+1}) using nominal data. In order to learn these transitions, we first generate training data with input-output pairs:

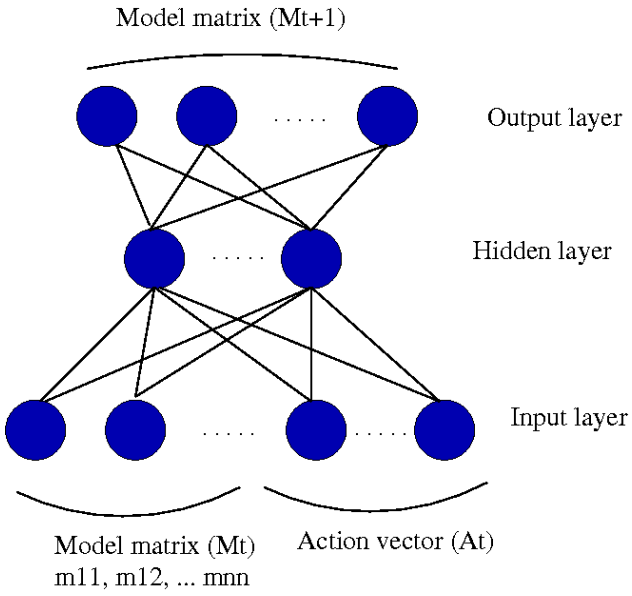


Figure 3. Neural Network model for transition $(M_t, a_t) \rightarrow M_{t+1}$

$(Input = (M_t, a_t),\ Output = M_{t+1})$. Since each particle in the particle filtering system is a sample of M_t, say $\tilde{M}$, we calculate the weight of a particle $\tilde{M}$ as the inverse of its error in predicting the observations between a_t and a_{t+1}, aggregated over the entire interval. Supposing that a_t occurs at time τ and a_{t+1} occurs at $\tau^{'} > \tau$, the error function is given by

$$e(\tilde{M}, \tau, \tau^{'}) = \sqrt{\sum_{k=\tau}^{k=\tau^{'}-1} \left\| \tilde{M}.\boldsymbol{z}_k - \boldsymbol{z}_{k+1} \right\|^2} \tag{3}$$

Thus particle filtering based localization produces a set of particles that predict the observations between a_t and a_{t+1} well, and the goodness of the prediction increases with an increasing number of frames between τ and $\tau^{'}$. However, this scheme also allows only one step for particle filtering between a_t and a_{t+1}, whereas particle filtering requires many steps to localize well. Therefore, we artificially create many steps by iteration. In order to allow for exploration of the search space, we apply noise to all of our particle models after each step.

Once the above process completes, we have (possibly many) samples of model matrices both before and after each action. Then for each successive action pair, a_t and a_{t+1}, we teach a neural network this transition $(M_t, a_t) \rightarrow M_{t+1}$. The neural network has $n^2 + a$ inputs – one for each element in the model matrix, and one for each action column– and n^2 output units to generate a model matrix, as shown in Figure 3. In order train this network, we need to map the "before" set of particles (i.e., samples of M_t) to the "after" set of particles (i.e., samples of M_{t+1}). One method for doing this would be to map all particles in the "before" set to the particle with the best weight in the "after" set. The problem with this method is that while we do assume that there is a consistent model between two actions, we do not always learn a single model with particle filtering but rather a distribution of models. Then we would like to map each "before" particle to the "after" particle that it is most likely to evolve into. In order to do this, we keep track of the ancestry of each particle while localizing such that each particle in the "after" set knows which particle it descended

from. Because of the resampling in particle filtering, many of the particles in the "before" set will have no representatives (descendants) in the "after" set. We resolve this by mapping each of these unrepresented particles to the represented particle that it most closely resembles. This leaves us with k groups of "before" particles with every particle in the "after" set being a descendant of one of these groups. Then, for each group, we select some particles - the amount we select is based on the size of the group relative to other groups - and train the neural network to transition each of those particles to the best-weighted descendant of the group.

V.C. Detecting Anomalies in a New Test

After we have learned a suitable transition model, we can use it to detect anomalies in a new test file. As in training we use particle filtering; however, we use it in an entirely different way. Where we used particle filtering to find the underlying model during training, here we mostly use it to make slight adjustments to the model to accommodate minor changes that can occur from run to run. Also, where we used entire intervals between actions as steps in the particle filtering process, here we use individual frames as steps. As before, we apply noise to the particle models after each step, but we substantially reduce the amount of noise to reduce exploration.

The testing process is as follows. A new file "A" can be tested against a specific nominal/training file "B". During the first segment (i.e., until a_1) in "A", we initialize the particle set with the set that the nominal file "B" reached at its first action by localization, when it was used to train the neural network. Then we localize in "A" using particle filtering to find the underlying model. When an action is reached, we use the neural network to transition each particle. We continue localizing (in a much less aggressive fashion, i.e., with low noise) and transitioning at actions. In this way, the model distribution during testing should resemble the nominal model distribution. At each frame, we log the average error (using the same

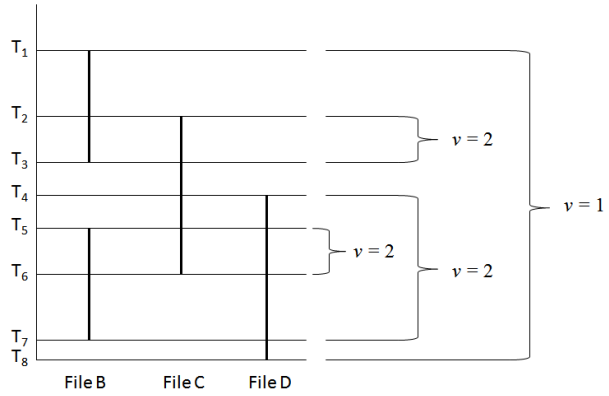


Figure 4. Illustration of the voting method used to output the final anomalies. The bars represent the timespans of sufficiently high errors for a given sensor, i.e, *alerts*, when compared to separate nominal runs, B, C and D. If the threshold is set to $v = 1$, the output is the timespan T_1–T_8, since at all times in this range at least one of B, C, or D produces an alert. If $v = 2$, the output is T_2–T_3 and T_4–T_7, since at other times fewer than 2 of B, C, or D produce an alert.

error function as in equation 3 with $\tau' = \tau + 1$) and the average error of each sensor (which breaks up equation 3 on sensor-by-sensor basis). If the current model distribution is unable to properly predict the sensor values, then the error will increase, and since the current model distribution should resemble the nominal distribution, this indicates an anomaly.

In order to extract the anomalies from test "A" against the nominal test "B", we require the sensor error logs from both "A" and "B". Note that this means we need to run the above test method on the nominal file "B" as well, to produce this data. We search for anomalies on a segment-by-segment basis. For each segment (a_t to a_{t+1}), we partition the corresponding segment in "A" and "B" into k blocks. Within each of these blocks, we find the average error for each sensor. If within a block, the average error for a particular sensor in "A" exceeds the corresponding value in the training log of "B" by some threshold t, then we produce an alert for that sensor over the time period covered by that block. To reduce the number of alerts reported, for each sensor, we combine alerts that occur in contiguous blocks.

Now it may be the case that multiple nominal files (i.e., many "B"s) are available, and we would like to

aggregate the test results on file "A" against *all nominal files*. We have devised a simple voting algorithm to accomplish this. First, we perform all the afformentioned steps for each nominal file, "B", so that we have a set of alerts for each. Then, for each sensor, we find timespans that are covered at all times by at least v alerts and report an anomaly for each. Figure 4 provides an example for a single sensor. In this figure, the vertical bars represent alerts report by nominal files, "B", "C", and "D". The brackets depict the alerts found for various values of v.

V.D. Experiments on a Given Data Set

We experimented with a dataset generated at the "E" test stand at Stennis Space Center. Test identifiers 63A–D were said to be nominal runs, while identifier 66A was known to have displayed an anomaly. Although 63A was redlined, the reason was later determined to be innocuous. Test 66A was known to have shown anomalous pressure reading at sensor ID 1491 (a pressure gauge). We first selected a set of 10 sensors using the method outlined in section V.A.1, but the output of this step did not include sensor ID 1491. Since we are able to detect *systemic anomalies* where an anomaly at one sensor is supposed to affect other sensors that are correlated with it, we needed the anomaly in sensor ID 1491 to reflect on other sensors. However, this anomaly appears to be isolated, and has not affected any other sensor. Therefore, we report the results from two alternative sensor sets: one that used only the 10 sensors selected in section V.A.1, and another that included sensor ID 1491 in addition to those 10 sensors.

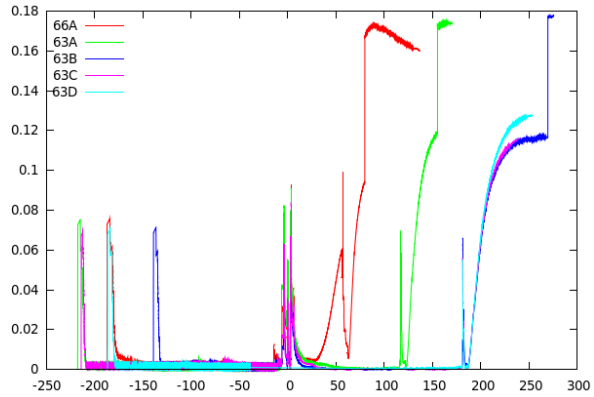


Figure 5. Anomalous Sensor (ID 1491) Included

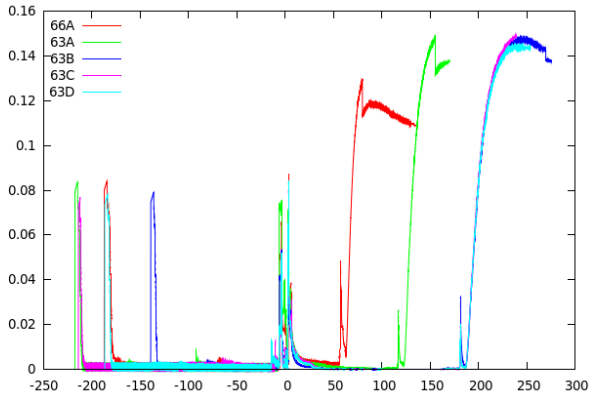


Figure 6. Anomalous Sensor (ID 1491) Excluded

Using all of 63A–D as nominal runs, we obtained the error plots of Figures 5 and 6 for tests 66A and 63A–D, for the cases that the sensor ID 1491 was included or excluded, respectively. There are many error peaks, but many of these occur at actions (i.e., model transition points), which also occur in the corresponding nominal runs, and are thus eliminated when compared to the latter. Figures 7 and 8 show the magnified section between $t = 0$ and $t = 70$, for the cases that sensor ID 1491 is selected or not, respectively. In Figure 7, we see a clear rise in the error in 66A from around $t = 30$ until the test redlined at $t = 55$. This demonstrates that our system is able to detect an anomaly well ahead of the redline, about the time of its actual onset. Figure 8 however, shows that if the anomalous sensor was not included, then the system fails to detect the anomaly, since the anomaly was non-systemic and limited to sensor 1491 alone.

Our system is also capable of weeding out false positives, and reporting only the names of the anomalous sensors and the timespan of each anomaly, by virtue of the voting procedure and limiting the duration of an anomaly to be at least 5 seconds. For the test 66A, the output of our system is the following:

```
Possible Anomaly:
-----Sensor Name:  PE_1491_C1c
-----Start Time:  41.7786
-----Duration:  14.1994
-----End Time:  55.978
```

Note that the voting procedure makes the anomaly undetectable until 41.78 secs, while Figure 7 reveals it earlier. This is the price paid for better sensitivity to false positives. The anomaly ends when the redline

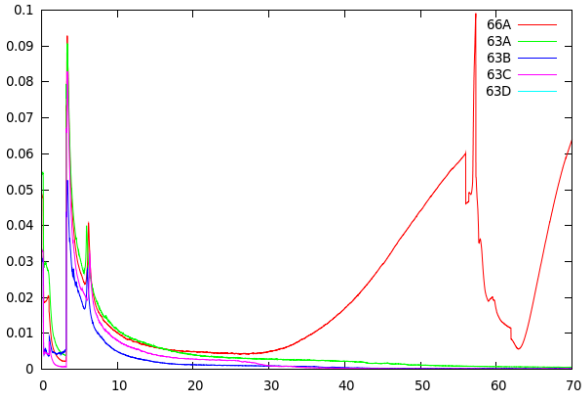


Figure 7. Anomalous Sensor (ID 1491) Included (Closer View)

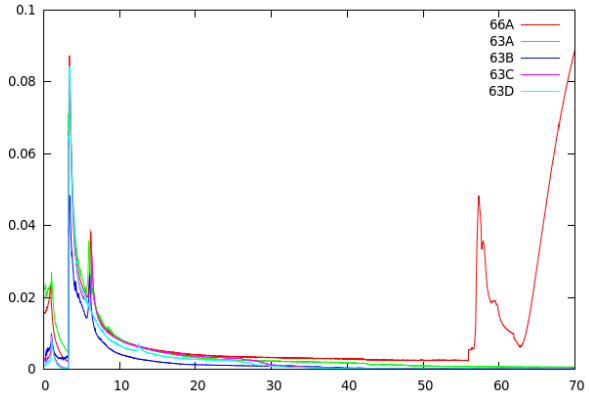


Figure 8. Anomalous Sensor (ID 1491) Excluded (Closer View)

actually happened in this test. No other test file, excepting the nominal run 63B shows any anomaly in the output. For 63B, the output is the following:

```
Possible Anomaly:
-----Sensor Name:  PE_9505_PWc
-----Start Time:  269.918
-----Duration:  5.915
-----End Time:  275.833
Possible Anomaly:
-----Sensor Name:  PE_1241_LOc
-----Start Time:  269.918
-----Duration:  5.915
-----End Time:  275.833
Possible Anomaly:
-----Sensor Name:  TC_1489_PWc
-----Start Time:  269.918
-----Duration:  5.915
-----End Time:  275.833
```

Two pressure gauges and a temperature gauge showed brief anomalous behaviors at the end of the test, during various purge steps. Since these occurred after the main test, and had brief durations, these can be overlooked. Hence, we observe that our approach shows good false positive performance.

VI. Prognosis of Anomalies

When an anomaly has been detected, the main role of prognosis is to project the impact of this anomaly onto *future* behavior, and identify other sensors that may indicate further trouble during the rest of the engine test. However, this must be done *before* further sensor data becomes available. There is a set of sensors that are identified as "redline" and "blueline", which must remain within certain pre-specified ranges during the test, and if a redline sensor falls outside its range then the test is automatically aborted. Sometimes, anomalies rising during a test are innocuous and aborting a test is unnecessary. Therefore, the ability to predict the likely outcome of an anomaly could prevent needless failures that are costly. If a set of sensors are predicted to redline in the future such that the pattern of failures indicates a major problem, then the test should be aborted. But if no other sensors are predicted to redline, or if the pattern of predicted failures are innocuous then the test might be continued.

The tracking method implemented so far could be an invaluable tool in the prediction of anomalies that could, in turn, be leading to redlines. Such prediction will not only raise the confidence in anomaly detection (by cross-verification with prediction, when the data becomes available), but also provide early warning for

redlines. The key insight is that there are temporal associations among the observed variables between the times when the anomalies occur, and the time of an ultimate redline, or even other anomalies later in the test. In the particle filtering framework, this can be roughly framed (with a subtle variation; see below) as the problem of estimating the posterior at a *future* time, $p(\boldsymbol{x}_{t+k}|\boldsymbol{z}_{1:t})$ where $k > 0$, based on observations until now. This posterior can be given by

$$p(\boldsymbol{x}_{t+k}|\boldsymbol{z}_{1:t}) = \int p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t}) \prod_{j=t+1}^{t+k} p(\boldsymbol{x}_j|\boldsymbol{x}_{j-1}) d\boldsymbol{x}_{t:t+k-1} \tag{4}$$

where the first factor in the integral is simply the posterior from the anomaly detection problem (update step, i.e., equation 1) as already described, $d\boldsymbol{x}_{t:t+k-1} = d\boldsymbol{x}_t.d\boldsymbol{x}_{t+1} \ldots d\boldsymbol{x}_{t+k-1}$, and $p(\boldsymbol{x}_j|\boldsymbol{x}_{j-1})$ could be the transition model already learned from *nominal* tests. Conceptually, equation 4 directly projects a current anomaly to future times to predict its effect on different parts in the system (spanned by particles). While the update equation (equation 1) helps detect a current anomaly, equation 4 helps predict the evolution of such an anomaly into a catastrophic failure possibly culminating in its abortion.

However, equation 4 is not trivially applicable. Because the transition model holds for a *nominal* run but not after an anomaly has occurred, the learned transition model *will not work for the anomalous sensors and the ones correlated to them*. Another reason why the learned transition models are actually unusable in prognosis, is as follows. The command plan in the rocket engine tests include a main test segment when the engine is firing, and many other commands/actions that either prepare for the test (e.g., chamber pressurizations), or finish up after the test (e.g., various purge actions), and the engine does not actually fire during these steps. The redline limits of various sensors are active only during the main segment, and not during the other activities. In fact, almost all sensors sit well outside their redline limits during these other segments. Therefore, while anomaly detection covers all activities during a test, prognosis of anomalies is only feasible during the main test segment (when the engine is firing) with no intervening actions. Therefore, equation 4 needs to be reinterpreted for prognosis.

A major difficulty with applying the particle filtering approach to prognostics is that the particles are frame-to-frame predictors, i.e., $\boldsymbol{x}_t$ is used to predict $\boldsymbol{z}_{t+1}$ (see equation 2). Such short-range predictions necessitate repeated applications of a particle to yield long-range predictions. However, this is problematic in the case of prognosis where the prediction cannot be matched against actual data, unlike the case with diagnosis. This is because the *actual data is not available* between the time an anomaly is diagnosed and the time that the prognosis must be made. Moreover, prognosis is often a *counter-factual operation*; while the actual test may have been aborted, we predict the outcome of allowing it to continue. With the actual data thus unavailable, we must rely on one step's prediction to feed into next step's input. This implies a (possibly long) chain of predictions, $\{\tilde{\boldsymbol{z}}_{k+1}, \tilde{\boldsymbol{z}}_{k+2}, \ldots\}$ (k being the last anomalous frame diagnosed), unrestrained by actual data through (re-)weightings, with possibly snowballing errors. We describe two approaches to prognosis, that implement equation 4 while addressing this problem.

VI.A. Pure Machine Learning Approach

First we took the set of all redline and blueline sensors as well as the 10 sensors selected in section V.A.1, and applied Algorithm 1 during the main test segment, separately for each nominal test run. This gives us a clustering of correlated sensors in this set during this segment. Then for each redline and blueline sensor, and for each nominal test run 63B–D, we trained a neural network to predict a sensor A's value at $t + \tau$, given the readings of the *other* sensors (i.e., excluding A unless A is in a singleton cluster) in A's cluster at t and the values t and τ as inputs. We used all integer values of t and τ from 1 to the length of the segment. Essentially, we assume that while the nominal models may not hold after an anomaly, the correlations among the sensors continue to hold and as a result, sensors correlated to an anomalous sensor may display anomalous behavior later. Therefore, this approach implements an alternative to equation 4, where the neural network is the substitute for the underlying model $\boldsymbol{x}$.

Suppose the main segment of a test run is between frames T' and T, and an anomaly is detected between k' and k, with X being the set of anomalous sensors. Therefore, the frames from $k + 1$ to T are effectively unavailable to prognosis. The sensors in X are linearly regressed from k' to k. Then for each redline and blueline sensor, we use the corresponding neural network to predict its value at T given its cluster-mates' values at k, and the values k and $T - k$ as inputs. If the predicted value falls outside the corresponding

(redline or blueline) range, then that sensor is flagged as a likely predicted anomaly. If sufficient nominal runs indicate anomaly on the same sensor, then it is flagged as a predicted anomaly by simple voting.

In the experimental data, most sensors give flat readings in the segment T' to T. Therefore, the neural networks simply learned to predict the same value for a sensor as the nominal training input, and the result of prognosis ended up being unreliable. For sanity check, we also ran prognosis on the nominal tests 63B–D, identifying a dummy anomalous segment k' to k but naming no anomalous sensor (i.e., $X = \emptyset$). The sensor predictions had high accuracies and no predicted anomaly was flagged in these nominal runs, as one would expect. However, even on the test run with the known anomaly 66A, the predictions matched the nominal runs and again no predicted anomaly was flagged. We observe that the outcome of prognosis in this particular data set may be acceptable, since the anomaly was known (in retrospect) to be caused by a clog in a sensor line rather than a system failure. Therefore no further anomaly was actually expected (although the test was automatically aborted since it was a redline sensor). But the fact that the neural nets failed to learn any useful pattern makes the outcome unreliable for other data.

VI.B. Particle Filtering Approach

With the machine learning approach being inconclusive, we attempted to directly implement equation 4, but with an eye to the possibility of snowballing errors. One solution to the short-range prediction problem is to purposely set the particles up for long-range predictions. For each choice of κ and each nominal test 63B–D, we localize a set of particles that give predictions κ frames into the future, i.e., $\tilde{\boldsymbol{z}}_{t+\kappa} = \boldsymbol{x}_{t,\kappa}.\tilde{\boldsymbol{z}}_t$, where $\boldsymbol{x}_{t,\kappa}$ are long-range predictor particles. This is achieved by applying particle filtering to sensor readings every κ frames, in each nominal run, for different choices of κ. In other words, we apply the following crude approximation of equation 4:

$$p(\boldsymbol{x}_{k+(T-k)}|\boldsymbol{z}_{1:k}) = p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}) \cdot p(\boldsymbol{x}_{k,k+\kappa_1}) \cdot p(\boldsymbol{x}_{k+\kappa_1,k+\kappa_1+\kappa_2}) \ldots \cdot p(\boldsymbol{x}_{k+\kappa_1+\ldots+\kappa_{y-1},T}) \tag{5}$$

where $\kappa_1 + \kappa_2 + \ldots + \kappa_y = T - k$, such that the chain of κs is as short as possible to minimize the impact of the independence assumption (i.e., the assumption that $p(\boldsymbol{x}_j|\boldsymbol{x}_{j-1}) = p(\boldsymbol{x}_j)$ within the main test segment, in equation 4). The symbols k, T were introduced in the previous subsection.

Given the length of 11046 adjusted frames in the main segment of the engine tests, we used $\kappa = 2500, 2000, 1500, 1000, 500, 250$ to be able to fit any length between the last frame of the anomaly and the end of the main test segment (analogous to the coin-changing problem) with an accuracy of within 250 frames (equivalent to ~ 4 secs of test time). In the actual test ID 66A, the anomaly ends at 55.978 secs (see section V.D), which corresponds to frame number $k = 3294$. From there to $T = 11046$, there are 7752 frames, and to cover these the fewest steps yield $\kappa_1 = \kappa_2 = \kappa_3 = 2500$, and $\kappa_4 = 250$, leaving out the last two frames (≈ 0.03 secs). The resulting predicted sensor readings $\tilde{\boldsymbol{z}}_T$ in test ID 66A flagged a gaseous hydrogen pressure sensor (ID 1481) as expected to redline at 135 secs after ignition (predicted by nominal test ID 63D), or at 179 secs after ignition (predicted by nominal tests ID 63B,C). We also ran a sanity check as described in the previous subsection, which flagged no predicted anomaly in the nominal tests. However, sensor ID 1481 appears to be unlikely to be affected by the anomalous sensor (ID 1491), since it is upstream from the latter, and could rather affect it instead of the other way around. We conclude that this prediction is an artifact of the correlation structure that particle filtering succeeds in capturing, but that the predicted anomaly is unlikely to materialize. Thus no further anomaly is expected, which supports the claim that this test could have been safely concluded and the redline was unnecessary. This was the conclusion of the test engineers, therefore this approach to prognosis appears to have succeeded, at least on the available data set.

VII. Conclusions and Future Work

We have presented a new approach to detect anomalies in sensor readings, and predict their future impact during an engine test. Our approach is model based; we employ machine learning to capture nominal behavior from known normal test runs, and then apply a non-parametric Bayesian filtering approach, viz., particle filtering, to detect anomalous behavior in a new test. Once an anomaly is detected, we apply an approximate smoothing approach (equation 5) to predict future anomalies in redline and blueline sensors to identify possibly impending shutdown. Our system was applied to a data set where one test contained a known anomaly that did abort the test but was later determined to be innocuous. The outcome is that our anomaly detection module successfully identified the known anomaly, and the prognosis module effectively

concluded that it was innocuous. Therefore, we conclude that if this system was operational during this test, an unnecessary (and costly) shutdown could have been averted.

Although our system was developed with careful consideration for general applicability to different engine test data sets, we were constrained by the availability of data sets. We were only able to test our system on one data set, and plan to test it on more data sets in the future.

VIII. Acknowledgments

This work was supported by NASA through grant number NNX08BA41A (under cooperative agreement with NASA John C. Stennis Space Center) and through a subcontract from The University of Mississippi under the terms of Agreement number NNG05GJ72H. The opinions expressed herein are those of the authors and do not necessarily reflect the views of NASA or The University of Mississippi.

References

[1]Schwabacher, M., "Machine Learning for Rocket Propulsion Health Monitoring," *SAE TRANSACTIONS*, Vol. 114, No. 1, 2005, pp. 1192–1197.

[2]Martin, R. A., Schwabacher, M., Oza, N. C., and Srivastava, A. N., "Comparison of Unsupervised Anomaly Detection Methods for Systems Health Management Using Space Shuttle Main Engine Data," *JANNAF Propulsion Meeting*, 2007.

[3]Guo, T. and Musgrave, J., "Neural network based sensor validation for reusable rocket engines," *Proceedings of the American Control Conference*, Vol. 2, 1995, pp. 1367–1372.

[4]Bradley, P., Mangasarian, O., and Street, W., "Clustering via Concave Minimization," *Advances in Neural Information Processing Systems*, edited by M. Mozer, M. Jordon, and T. Petsche, MIT Press, 1997, pp. 368–374.

[5]Iverson, D. L., "Inductive System Health Monitoring," *Proceedings of The 2004 International Conference on Artificial Intelligence*, Las Vegas, NV, 2004.

[6]Agogino, A. and Tumer, K., "Entropy based anomaly detection applied to space shuttle main engines," *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, 2006.

[7]Martin, R. A., "Unsupervised anomaly detection and diagnosis for liquid rocket engine propulsion," *Proceedings of the IEEE Aerospace Conference*, Big Sky, MT, 2007.