Online Learning of Spacecraft Simulation Models

Justin Thomas*
United Space Alliance
600 Gemini
Houston, TX 77058

Christoph Eick
University of Houston
4800 Calhoun Road
Houston, TX 77204

Abstract

Spacecraft simulation is an integral part of NASA mission planning, real-time mission support, training, and systems engineering. Existing approaches that power these simulations cannot quickly react to the dynamic and complex behavior of the International Space Station (ISS). To address this problem, this paper introduces a unique and efficient method for continuously learning highly accurate models from real-time streaming sensor data, relying on an online learning approach. This approach revolutionizes NASA simulation techniques for space missions by providing models that quickly adapt to real-world feedback without human intervention. A novel *regional sliding-window* technique for online learning of simulation models is proposed that regionally maintains the most recent data. We also explore a knowledge fusion approach to reduce predictive error spikes when confronted with making predictions in situations that are quite different from training scenarios. We demonstrate substantial error reductions up to 76% in our experimental evaluation on the ISS Electrical Power System, discuss the successful deployment of our software in the ISS Mission Control Center for ground-based simulations, and outline future adoption at NASA.

Introduction

Simulation plays a crucial role in NASA spaceflight. Software models recreate hardware behavior when using the real system is impossible due to costs, safety, or operational constraints. Spacecraft simulation is an integral part of mission planning, real-time mission support, training, and systems engineering.

Common practice at NASA in generating simulation models is to use hardware specifications, manufacturer test data, and physics to derive equation systems that describe system behavior. The resulting equations may range from simple algebraic expressions to complex integrals and differential equations that require advanced numerical analysis techniques. The final software implementations of these equation systems are known as *engineering models*. After initial model construction, engineers must tediously evaluate and adjust models in order to match true system behavior (Jannette et al. 2002). Engineering model adjustments range from modifying coefficients to changing equation forms.

With the sheer size and evolving nature of the International Space Station, engineers cannot constantly adjust models to reflect current system behavior due to schedule and budgetary constraints. In this paper, we present a solution that uniquely solves this challenging problem by using machine learning to continuously and autonomously construct highly accurate models from real-time *telemetry* (streaming sensor data).

This solution advances the state of the art in NASA simulation techniques that traditionally require manual model construction and adaptation. The work in (Bay, Shapiro, and Langley 2002) recognizes the need for a machine learning solution to this problem, but fails to address key problems such as model construction without the restrictions of specific equation forms, the use of supplemental knowledge during extrapolation, and efficient online system operation. Our solution examines a drastically different approach that does not require any assumptions about the mathematical structure of the model, uses a model fusion approach to address extrapolation, and efficiently reacts to changes in system behavior within seconds.

Our approach pre-processes spacecraft telemetry, maintains a constant-size training dataset, and employs a regional sliding-window that preserves recent examples for localized regions within the input space, allowing for accurate predictions across all valid regions of the input space.

Using the resulting representative training dataset, we evaluate a set of candidate algorithms and learn a simulation model using the best training algorithm. Each resulting model is sent to the spacecraft simulator in real-time to match current system behavior. Since the simulator can request predictions for regions of the input space not contained in our training dataset, we employ a model fusion approach that uses the knowledge contained in existing engineering models to eliminate large spikes in error during drastic extrapolation.

The unique contributions of the paper include:

- Online autonomous learning of highly accurate spacecraft simulation models from real-time telemetry
- Regional sliding-windows that outperform traditional sliding-windows for simulation model construction

*Now at the Johns Hopkins University Applied Physics Laboratory

Figure 1: Johnson Space Center Mission Control Center (photo courtesy NASA)

- Model fusion that additionally uses traditional engineering models to make more reliable predictions in situations that are quite different from training scenarios
- A new system architecture that integrates machine learning into the simulation model construction and adaptation process

We evaluate our technique using the ISS Electrical Power System (EPS) and NASA historical data archives of spacecraft telemetry. Evaluation results demonstrate significant accuracy improvements over existing EPS engineering models.

The paper is organized as follows: The Background section provides relevant background information, the Technical Approach section describes our solution in detail, the Experimental Results section offers results from our experimental evaluation, the Deployment section discusses the deployment of our technique at NASA, and finally, the Conclusion and the Future Work sections conclude with a summary and identification of future work.

Background

Spacecraft Simulation

For ISS mission operations (Figure 1), spacecraft system simulations assist flight controllers and engineers with preflight planning, in-flight monitoring, real-time mission planning, and post-flight analysis. Simulations allow engineers to answer critical questions such as the following: How long until the crew exhausts the oxygen supply during a cabin pressure leak? How can we orient the spacecraft to retain communication when mechanical antenna positioning fails? Is enough power available to run a science rack experiment for the required fourteen days?

Engineers generate initial models using hardware specifications, manufacturer test data, and physics to derive a set of equations that describe system behavior. The resulting equations range from simple algebraic expressions to complex integrals and differential equations that require advanced numerical analysis techniques. The level of fidelity is dictated by accuracy requirements.

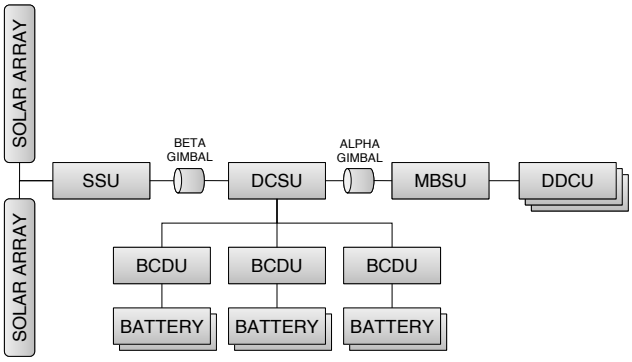

Figure 2: ISS Electrical Power System schematic

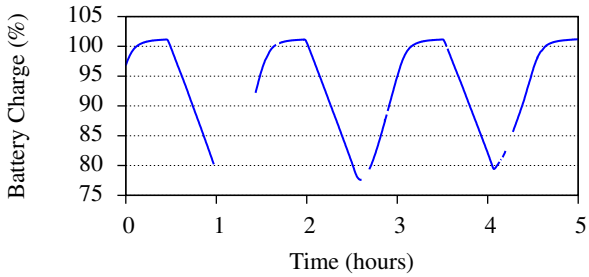

Figure 3: Battery charge level telemetry

After initial model construction, engineers must tediously evaluate and adjust models in order to match true system behavior. Engineering model adjustments range from minor changes of coefficients to major refinements to equation forms.

Simulation models need to reflect real-world system behavior, otherwise their value is limited. Engineers find it difficult and time-consuming to create and maintain accurate models for spacecraft systems for the following reasons:

- Human-rated spacecraft systems are complex
- True performance is only observable when the system is deployed in space
- Abnormal scenarios are difficult to understand
- System operation can evolve over time
- Extensive human involvement is necessary to adapt and refine models from system feedback

We address these challenges by autonomously generating highly accurate models that always reflect current system behavior.

ISS Electrical Power System

We chose the ISS EPS (Gietl et al. 2000) to evaluate our technical approach due to our domain knowledge of the EPS. However, our approach is generic and not tailored to any specific hardware system.

Our focus was on two core components: the battery charge/discharge unit (BCDU) and the battery itself. The hardware layout for a single power channel (eight total) is illustrated in Figure 2. The ISS absorbs sunlight through its massive solar arrays and converts the energy into a usable power source through a network of complex electrical equipment. An array of nickel-hydrogen batteries stores excess energy for use during orbital eclipse. The ISS orbit results in cyclic battery charge/discharge behavior due to the periodic transition from eclipse to insolation (periods of solar radiation) as demonstrated in Figure 3.

An EPS model used for power resource planning must predict future performance in a temporal manner. This forecasting requirement significantly complicates the problem. An EPS model must forecast battery charge profiles, as illustrated in Figure 3, given only initial conditions and a time-series of predicted inputs. Flight planners need to know if power will be available for the next week, not simply the immediate future.

Technical Approach

In the Requirement Analysis section, we provide a requirement analysis for our solution. Then in the System Architecture section, we outline the system architecture and illustrate the flow of information from incoming telemetry to final simulation output. We then focus on the following solutions necessary to solve specific problems: regional sliding-windows (Regional Sliding-Window section), model learning and evaluation (Model Learning and Evaluation section), and model fusion (Model Fusion section).

Requirement Analysis

We must solve a series of problems in order to effectively learn spacecraft simulation models in an online fashion:

- Process large amounts of continuously streaming sensor data
- Create models that forecast into the future
- Account for both frequent and long periods of missing data
- Create models capable of making predictions for any valid region of the input space
- Adjust models for system degradation[1]
- Learn and evaluate models without human intervention

We will discuss these requirements in detail throughout the discussion of our approach.

System Architecture

Our architecture consists of the offline analysis components and online learning components as shown in Figure 4. We now describe the flow of data through these system components. Before running our system, the Feature Selection component determines the optimal set of sensors that generate the best possible model. Once our system is running,

[1]Some EPS devices degrade over time or when operated past normal conditions.

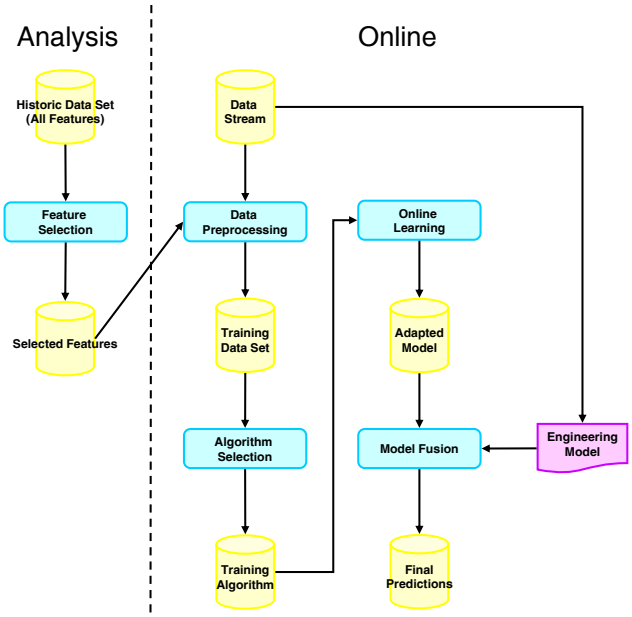

Figure 4: System architecture

Algorithm 1 Regional Sliding-Window Algorithm

Input: trainingSet, dataStream $\vec{x}[]$, size n, size k
Initialize $trainingSet \Leftarrow \emptyset$
loop
  $newPoints \Leftarrow$ next n samples from x
  $trainingSet \Leftarrow trainingSet \cup newPoints$
  $trainingSet \Leftarrow k$-means from $trainingSet$
end loop

all incoming telemetry runs through the Data Preprocessing component to condition the data appropriately for model learning and evaluation. At a fixed frequency, the system will revise the training dataset using the regional sliding-window. The system sends this training dataset to the Algorithm Selection component where the best algorithm is selected from a set of candidate training algorithms. The Online Learning component then learns a system model using the current training set. The system then sends the resulting *learned models* to the simulator where the Model Fusion component combines the knowledge contained in existing engineering models to prevent highly inaccurate predictions.

Detailed descriptions of the Feature Selection and Data Preprocessing components are provided in (Thomas 2007) and are not the focus of this paper.

Regional Sliding-Window

When learning from data streams (Babcock et al. 2002; Gaber, Zaslavsky, and Krishnaswamy 2005), it is not realistic to use the entire data stream for model learning. In our case, a year of operational data sampled at 10HZ results in a training set with over 3 million data points (1 GB). A popular technique to handle data streams is to only retain the n most recent data points. This is known as a sliding-window (Wid-

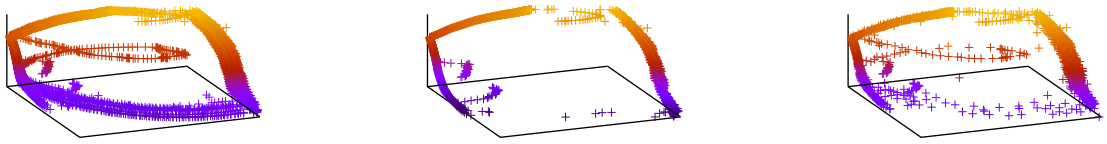

Figure 5: Regional sliding-window example – A full day of battery data (left) shows a loop in the middle of the input space. This loop is lost the next day using a traditional sliding-window (middle). However, a regional sliding-window maintains this important knowledge (right).

mer and Kubat 1996). This simplistic model works well in many situations, but suffers from the drawback that knowledge is lost when it leaves the sliding-window (Figure 5). For a spacecraft simulation model, we need examples covering the entire input space or we risk wildly erroneous predictions during inductive model extrapolation. We must preserve this knowledge as it unpredictably arrives in telemetry. We propose a regional sliding-window for learning accurate simulation models.

We use k-means clustering to create a representative dataset across all exercised operational regions. Algorithm 1 takes an initial training dataset and appends n samples from data stream $\vec{x}[]$. Then k-means is used to generate a large number of representatives, keeping the training set a constant size k. Examples in the training set that are nearby the new samples from $\vec{x}[]$ will be shifted towards the new data as k-means determines the new representatives. Examples from the training set that are not nearby the new data will not move. We desire this behavior since we do not want to discount or remove elder data that are most recent for a particular region in the input space. By selecting a sufficiently large value for k, we can maintain data points across the entire input space as shown in Figure 5.

The selection of the k and n parameters are important in the performance of this regional sliding window approach. Our experimental evaluations found that n must be much smaller than k (typically around 90-95% smaller). The value of k is related to the number of data points necessary to preserve valid operating values across the input space. We experimentally determined k by producing a series of plots on a data set with increasing values of k, and then increasing the value of k by 20% to account for future data not included in our input dataset. If k is too large, our runtime performance will suffer due to increased dataset size during training, and if k is too small, then we might not be able to preserve enough data for quality training. Also n controls the training frequency, since after n input points are buffered, we then execute k-means to perform a reclustering. As the value of n decreases, our system performance will decrease.

Many system models must also account for system degradation. In the field of machine learning, this is known as the concept drift problem (Widmer and Kubat 1996). Concept drift is best described as dynamic change to the underlying data-generating distribution. In our domain, the EPS components tend to degrade over time, resulting in changes to the system behavior and corresponding output. Our regional sliding-window technique also indirectly accounts for the concept drift phenomenon since incoming data points will influence the elder training examples in that local region.

Model Learning and Evaluation

When learning from data streams, our system takes in the regional sliding-window of telemetry and generates a model at a fixed frequency. During the learning process, the system selects the best algorithm from a candidate set. We selected the candidate algorithms by examining our problem characteristics (real-valued inputs/outputs, efficient prediction performance, etc…) and experimenting on sample datasets. We used the following algorithms in our experiments:

- Artificial Neural Networks (single and boosted ensemble)
- Linear Regression
- Regression Trees (single and bagged ensemble)
- Model Trees (single and boosted ensemble) (Quinlan 1992)

The system selects the algorithm that generates system models with the lowest mean absolute error on an independent test set. We will discuss the performance of each algorithm in our experimental results (Experimental Results section).

The EPS telemetry shown in Figure 3 contains significant gaps which represent data unavailability due to communication outages between the ISS and the MCC. The Tracking and Data Relay Satellite (TDRS) communications system is a shared resource and we expect such communication outages. In the extreme case, data can be unavailable for up to 30 minutes.

For time-independent models (such as the BCDU model), we construct training sets and independent model evaluation sets by simply removing all data points with missing data from the full dataset. The occurrence of missing data is completely independent of EPS system operation and we have ample training data.

For time-dependent forecasting models (such as the battery model), we create a *lagged variable* (Chatfield and Weigend 1994) for the time-delayed output. To create a

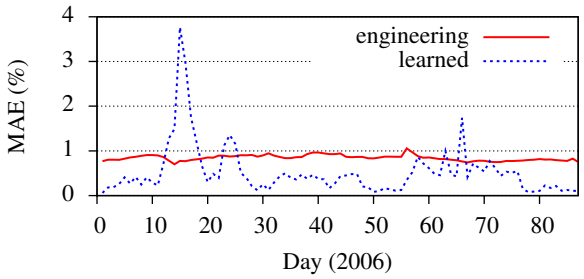

Figure 6: Battery model error comparison (engineering model vs. learned model)

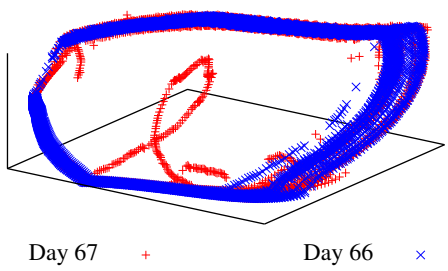

Figure 7: Previously unobserved battery data points

lagged variable, the system creates a new feature for the data point at time t with the output from the data point at time $t+1$.

To evaluate forecasting models, we cannot use traditional n-fold cross validation since data points are now interdependent. We must evaluate our models using a full time-ordered sequence of independent test data. For our evaluations, we desire a dataset without missing data for at least 90 minutes (one ISS orbit), but these datasets are extremely scarce. Instead we used linear interpolation to complete missing values. To avoid overly biasing the evaluation data, we experimentally determined the maximum duration of missing data that meets an externally defined quality threshold. For our evaluations, we generated ample test data using a missing data threshold of 40 seconds, well under the experimentally determined maximum threshold of 100 seconds.

Model Fusion

Simulation models provide engineers a means of experimenting and trying out many potential scenarios. This means system models must be able to make predictions in any valid region of the input space at any time. We must account for cases when our system model must predict in a region in the input space where no training examples exist. This is the extrapolation problem that all inductive algorithms face.

Model fusion is a knowledge fusion approach that uses traditional engineering models to supplement the learned models. Our software uses the output from the learned model when the incoming data point is very similar to the training dataset, otherwise we use the output from the engineering model. We observed that engineering models provide useful background knowledge to reduce predictive error spikes when confronted with making predictions in situations that are quite different from the training scenarios used when learning the machine learning model. We refer to the resulting model as the *fused model*.

As we show in Figure 6, battery models created using our online learning technique outperform the existing engineering model with the exception of a few large spikes. Visual inspection of the telemetry data in Figure 7 shows that during one of these spikes, there were novel examples reflected by the loop in the middle of the normal operating region. This scenario demonstrates the need for model fusion since the learned model did not accurately predict these inputs.

Algorithm 2 Model Fusion Algorithm

Input: dataPoint, means[], covars[], threshold
$c \Leftarrow NearestCluster(dataPoint, means, covars)$
$distance \Leftarrow MahalanobisDistance(dataPoint, c)$
if $distance > threshold$ **then**
  $output \Leftarrow EngineeringModelOutput(dataPoint)$
else
  $output \Leftarrow LearnedModelOutput(dataPoint)$
end if

To determine if the incoming data point is similar to the training dataset, we turn to unsupervised density estimation techniques as proposed by Bishop (Bishop Aug 1994). We use a Gaussian Mixture Model (GMM) density estimator to determine if the new data point is novel when compared to the training dataset. Our software achieves this by measuring the distance from the clusters generated by the standard Expectation-Maximization training procedure as shown in Algorithm 2. If the Mahalanobis distance is greater than our global novelty threshold, the incoming data point is deemed novel and we use the output from the engineering model. The global threshold is set so that 99% of the training set is deemed nominal to allow for some noise.

We use the Mahalanobis distance function rather than the normal probability density function to meet simulator runtime performance requirements. The software calls the model output generation function at such a high rate that any increase in run-time performance has drastic effects on overall simulation time. The probability density function uses several exponential terms that have a measurable impact on performance.

Our initial approach relied on a k nearest neighbor for determining the novelty of incoming data points. This approach was extremely slow due to the linear search time when generating each model output. We deemed this approach unusable due to the unacceptable performance although initial investigations into kd-trees and ball-trees offered some hope.

Figure 8 shows the large error spikes by the learned model in Figure 6 are reduced in the final fused model. For a few

Approach	Battery	BCDU Current	BCDU Voltage
Existing engineering model	0.00831	1.1305	4.8653
Learned model with traditional sliding-window	0.01096	0.3037	0.8237
Learned model with regional sliding-window	0.00513	0.2273	0.6538
Final model (model fusion and regional sliding-window)	0.00623	0.3079	1.2574

Table 1: Results summary (mean absolute errors)

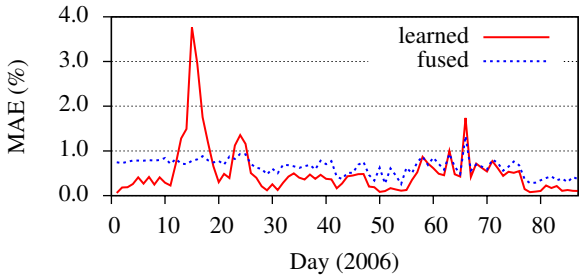

Figure 8: Battery model error comparison (learned model vs. fused model)

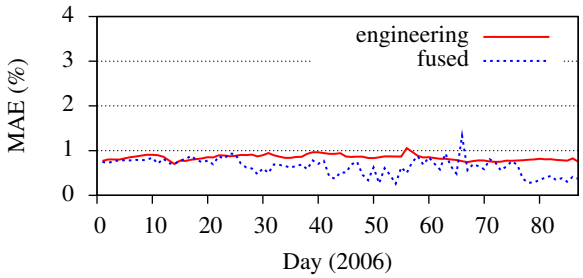

Figure 9: Battery model error comparison (engineering model vs. fused model)

datasets, the mean error in the final fused model is larger than the learned model, but these small sacrifices are necessary to bound large error spikes caused by extreme extrapolation.

Model fusion provides a safety net to ensure that adaptive model extrapolation does not introduce bizarre results. Unusual output will hurt the users confidence and decrease their trust in this intelligent system. This safeguard is necessary for real-world adoption in spacecraft simulation.

Experimental Results

For our experimental evaluation, we used 90 days of ISS historical telemetry data from year 2006. First, we performed feature selection for the battery and BCDU devices. Next, we started a regional sliding-window with k=2000 and n=100 to generate training datasets. Then, our software learned new models every 24 hours. Finally, all models were evaluated on independent test sets one week into the future.

A summary of the results is provided in Table 1. The regional sliding-window approach improves over traditional sliding-windows by reducing mean absolute error (MAE) by 53% (battery), 25% (BCDU current), and 21% (BCDU voltage). Model fusion slightly increases MAE for all learned models, but removes drastic spikes in error by reducing maximum errors by 70% (battery), 2% (BCDU current), and 73% (BCDU voltage). Our final models improve over the existing engineering models with 25% (battery), 73% (BCDU current), and 74% (BCDU voltage) reductions in MAE. In all cases, the system selected boosted model trees as the learning algorithm.

The low level results for each 24 hour period are provided in Figure 9 (battery), Figure 10 (BCDU current), and Figure 11 (BCDU voltage). These figures compare the final fused model against the existing engineering model.

Figure 12 provides an example of the final fused battery

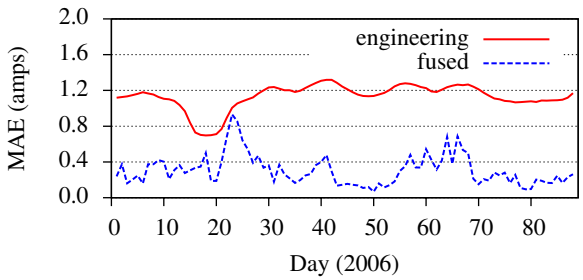

Figure 10: BCDU current model error comparison (engineering model vs. fused model)

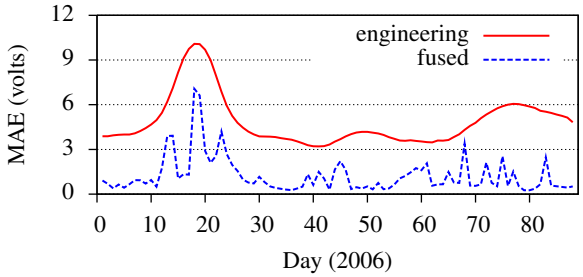

Figure 11: BCDU voltage model error comparison (engineering model vs. fused model)

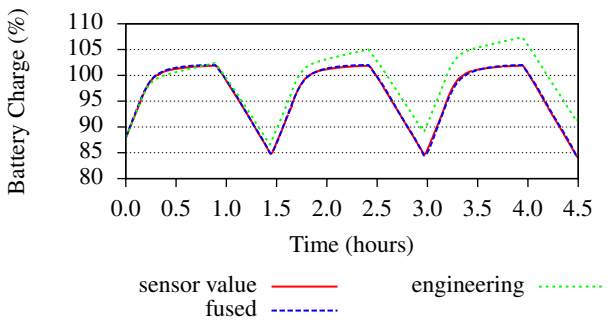

Figure 12: Battery model output comparison (engineering model vs. fused model)

model output compared against the true sensor value and the existing engineering model output over a 4.5 hour period. The learned model more accurately simulates true EPS battery performance, while the engineering model slowly drifts away.

Deployment

ISS power system flight controllers approved our proposal to integrate this method into a ground-based EPS model used for ISS power planning in the Mission Control Center. This is the only method in use at the NASA Johnson Space Center that applies machine learning to spacecraft system simulation. We split this project into two phases in order to evaluate this method in stages and build user confidence in machine learning techniques for critical mission support.

The first phase integrated learned models generated offline from historical data. Models are not continuously learned and updated within the simulator. Without full online adaptation, we must manually trigger the learning process to generate new models from the most recent telemetry data. This allows human validation of new models, but does not meet our goal of fully adaptive and automated model learning.

The first phase completed in April 2008 with deployment into the Solar software application (Thomas and Downing 2008) for use in ISS power planning. The learned models help to increase accuracy for a critical constraints-based solar array planning problem (Thomas and Downing 2008). This increase is exemplified by comparisons in the Experimental Results section above. We also received positive feedback from the flight controllers who use Solar, but the exact accuracy increases are hard to quantify because the model predicts the amount of power available on the ISS; a number that cannot be measured under normal circumstances because the ISS must not operate at maximum power consumption levels.

The existing EPS model was implemented in Java, and therefore we also implemented the software supporting online learning of spacecraft simulation models in Java. Our software currently runs within a Windows environment on Intel x86-based laptops, although the Java implementation allows for portability across a variety of platforms. Total project effort was estimated at 500 hours from problem description to deployed software (first phase). We leveraged the WEKA data mining software library (Witten and Frank 2005) for the Model Tree and k-means implementation, however we implemented our own data preprocessing, results evaluation, and Gaussian Mixture Model algorithm due to custom project requirements. Such software reuse helps to reduce development costs.

The second phase consists of implementing the on-line approach provided in this paper. Software will continuously generate models from real-time telemetry and autonomously update the simulation. Our model evaluation technique will collect statistics and validation test sets to perform trustworthy automated model validation. The second phase is under consideration for 2009.

Conclusion

This paper introduced a unique and efficient method for continuously learning highly accurate models from real-time streaming sensor data, relying on an online learning approach. This approach revolutionizes NASA simulation techniques for space missions by providing models that quickly adapt to real-world feedback without human intervention. A novel regional sliding-window technique for online learning of simulation models was proposed that regionally maintains the most recent data. We also explored a knowledge fusion approach to reduce predictive error spikes when confronted with making predictions in situations that are quite different from training scenarios. We demonstrated substantial error reductions up to 76% in our experimental evaluation on the ISS Electrical Power System which resulted in our approach being the only application of machine learning in spacecraft simulation in use at the NASA Johnson Space Center.

Future Work

We plan to explore additional approaches for the regional sliding-window technique to improve upon our current k-means technique. We also plan to explore online feature selection to dynamically add and remove sensors from the training set.

Future applications of this technology include, but are not limited to, more ISS systems, more NASA simulation facilities, and other NASA vehicles such as the new Orion spacecraft. We plan to design and develop a run-time architecture and software tool suite to allow engineers to build simulations using adaptive learning techniques.

Acknowledgments

We would like to thank the reviewers, Susan Ahrens, Natalie Ducote, Ray Halyard, Mike Genest, Jon Yemin, Patrick Walter, and the PRO and PHALCON flight control groups for their participation and contributions. This work was funded by NASA Johnson Space Center under contract NAS9-20000.

References

Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; and Widom, J. 2002. Models and issues in data stream systems. In *PODS '02*, 1–16. New York, NY, USA: ACM.

Bay, S. D.; Shapiro, D. G.; and Langley, P. 2002. Revising engineering models: Combining computational discovery with knowledge. In *ECML '02*, 10–22. London, UK: Springer-Verlag.

Bishop, C. Aug 1994. Novelty detection and neural network validation. *Vision, Image and Signal Processing, IEE Proceedings* 141(4):217–222.

Chatfield, C., and Weigend, A. S. 1994. Time series prediction: Forecasting the future and understanding the past. *International Journal of Forecasting* 10(1):161–163.

Gaber, M. M.; Zaslavsky, A.; and Krishnaswamy, S. 2005. Mining data streams: a review. *SIGMOD Rec.* 34(2):18–26.

Gietl, E. B.; Gholdson, E. W.; Manners, B. A.; and Delventhal, R. A. 2000. The electric power system of the international space station - a platform for power technology development. Technical Report TM-2000-210209, NASA.

Jannette, A. G.; Hojnicki, J. S.; McKissock, D. B.; Fincannon, J.; Kerslake, T. W.; and Rodriguez, C. D. 2002. Validation of international space station electrical performance model via on-orbit telemetry. In *ECEC '02*, 45–50.

Quinlan, J. R. 1992. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, 343–348.

Thomas, J., and Downing, N. 2008. A flexible spacecraft operations strategy for complex constraint management. In *AIAA SpaceOps 2008*.

Thomas, J. 2007. Adaptive modeling of the international space station electrical power system. Master's thesis, University of Houston, Department of Computer Science.

Widmer, G., and Kubat, M. 1996. Learning in the presence of concept drift and hidden contexts. *Machine Learning* 23(1):69–101.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical machine learning tools and techniques*. San Francisco, CA: Morgan Kaufmann, 2nd edition.