Toward an Automated Signature Recognition Toolkit for Mission Operations

T. Cleghorn (NASA/JSC) C. Culbert (NASA/JSC) D. Hammen†(MITRE)
P. Laird* (NASA/ARC) M. Macha (NASA/JSC) T. Moebes (SAIC)
L. Perrine (NASA/JSC) R. Saul (NASA/RECOM) R. Shelton† (NASA/JSC)

KEY WORDS AND PHRASES

Event detection, pattern recognition, signature recognition, telemetry, time series.

SIGNATURE RECOGNITION

Signature recognition is the problem of identifying an event or events from its time series. The generic problem has numerous applications to science and engineering. At NASA's Johnson Space Center, for example, mission control personnel, using electronic displays and strip chart recorders, monitor telemetry data from three-phase electrical buses on the Space Shuttle and maintain records of device activation and disactivation. Since few electrical devices have sensors to indicate their actual status, changes of state are inferred from characteristic current and voltage fluctuations. Controllers recognize these events both by examining the waveform signatures and by listening to audio channels between ground and crew. Recently the authors have developed a prototype system that identifies major electrical events from the telemetry and displays them on a workstation. Eventually the system will be able to identify accurately the signatures of over fifty distinct events in real time, while contending with noise, intermittent loss of signal, overlapping events, and other complications. This system is just one of many possible signature recognition applications in Mission Control. While much of the technology underlying these applications is the same, each application has unique data characteristics, and every control position has its own interface and performance requirements. There is a need, therefore, for CASE tools that can reduce the time to implement a running signature recognition application from months to weeks or days. This paper describes our work to date and our future plans.

*NASA Ames Research Center, Moffett Field, CA. 94035-1000 (USA). Email: LAIRD@PTOLEMY.ARC.NASA.GOV

†NASA Johnson Space Flight Center, PT4, 2102 NASA Road 1, Houston, TX 77058-3696 (USA). Email: SHELTON@GOTHAMCITY.JSC.NASA.GOV, DHAMMEN@MITRE.ORG

DEVELOPING A SIGNATURE RECOGNITION APPLICATION

A typical signature-recognition application monitors a data stream and is activated by an "event," as defined by the satisfaction of certain conditions. Data is then taken from the data stream, filtered and converted, and passed to a pattern-recognition module. The module decides to what class the event belongs and adjusts the controller's display. The event may also be captured for later offline use.
The following six steps are followed in designing and implementing a signature recognition application:

1. *Identify the users.* At Mission Control the end users (and the domain experts) are mission controllers.
2. *Acquire the data.* Training the system to identify signatures requires that one collect a set of correctly labeled signatures. Other information in the form of rules may also be required. This data is usually in short supply, either because some events occur rarely (e.g., engine failures) or because accurately labeled events are unavailable in machine-readable form. Ensuring the accuracy of the training data is, of course, critical.
3. *Design the pattern-recognition method(s).* Along with classical pattern recognition (PR) methods, more general techniques like neural networks, genetic algorithms, and

decision trees are effective and easy to understand. User confidence in the PR method is very important: for our users to accept the application, they need (and want) to understand the PR method conceptually, and are unwilling to base decisions upon an inscrutable answer from a "black box."

4. *Design the user interface.* Ideally the user interface should be an integral part of the system design from inception. Since a certain amount of experimentation is needed to ascertain the best presentation, a flexible interface tool for rapid prototyping is invaluable.
5. *Engineer the system architecture.* Online data typically flow from the input line, through various filters and formatting routines, onto and off of queues, to pattern recognizers, screen displays, and archival storage. Ensuring that the system can keep pace with this flow is essential.
6. *Evaluate the results.* One must plan to monitor the accuracy and performance of the running system over time, because the environment is constantly changing and the signatures with it.

THE SIGNATURE RECOGNITION TOOLKIT CONCEPT

Our goal is to automate the above steps to the extent possible, and to place much of the specification, implementation, and maintenance tasks into the hands of the end users. Current application development environments like AVS, Khoros, Matlab, etc., are useful for prototyping but do not produce a real-time application. Naturally, however, we borrow many ideas from these existing toolkits.

The task of enlisting the users is, of course, inherently human, so automation begins with the data acquisition step. At Johnson Space Center's Mission Control, flexible subsystems are in place that distribute telemetry data to the applications. In order to apply pattern recognition to this stream, we must identify repeatable event instances in the available data that can then be subjected to pattern analysis. Data segmentation—extracting finite events from the stream—can be very subtle owing to noise and other unforeseen properties. Alternately, one can monitor the stream continuously, treating every data sample as an event; but when the sample rate is high, performance requirements will severely restrict the possible analysis.

A "Data Warehouse" (DW) tool that runs offline can capture signatures in a database, display them for for domain experts to examine and label, and later format them as input to training programs. The same tool can record rule-based knowledge from the experts and, later in the process, help with system performance monitoring (see below).

The third step (designing a PR technique) can be substantially automated, but will often entail some assistance from an expert. Any good toolbox contains multi-purpose neural network, decision tree, and genetic algorithm software, as well as more specialized techniques. But there are so many problem-specific issues—e.g., the amount and kind of generalization, measures of accuracy and confidence, tradeoffs between speed and power, noise compensation, feature extraction, training time versus recognition time, and allowance for future growth in training data and the number of classification labels—that we believe that the support of a PR engineer will be required.

Step four is greatly simplified by today's interface building tools. Connecting the interface widgets to the data stream is straightforward except for the task of ensuring that dataflow bottlenecks do not lose input data. This task may require the assistance of a software engineer. The time to accomplish this task can be mitigated if the toolbox modules are fitted with calling interfaces so that they can be "plugged into" one another without total recompilation, much like the components on an electronic breadboard.

Finally, part of ongoing performance monitoring includes the task of having the users validate the labels assigned by the system, and using the results to check that the accuracy of the system does not degrade. We find that classifiers often need to be retrained. The DW tool can archive the online events with the system-assigned labels, collect the results of user validation, calculate and report the accuracy, and automatically retrain the classifiers on the most current samples.

In summary, a signature-application toolkit will contain the following software components integrated into a uniform environment:

- A mechanism for capturing data and segmenting signature events.
- A DataWarehouse tool that saves labeled events for training and testing and formats them in various ways for output to software components. Later this same tool supports the process of monitoring the performance

and accuracy of the system over time.

- A library of PR modules that can be trained to classify events to specified accuracy and confidence levels.
- An Interface Builder so that end users can design and maintain their view of the events as they occur.
- A library of dataflow components equipped with a flexible module-to-module interface, so that the system can be assembled simply by describing the modules and their connections.

Given this, the users will still need a PR engineer to define events and evaluate the PR options, and a software engineer to assemble and debug the system.

STATUS OF THE SIGNATURE RECOGNITION TOOLKIT

This description comes mostly from our experiences constructing prototypes in two domains. Initial work has begun on a third domain, and plans are to build several more prototypes or pre-prototypes in order to converge on a toolkit specification and design.

Implemented applications.

The two implemented domains are nearly opposites. One ("EGIL") entails recognizing about fifty types of events of several seconds' duration that occur regularly during the mission. Since unseen (unlabeled) events also occur, the classifiers must include a "none-of-the-above" category—a requirement that makes the recognition task much more challenging. Additional complications occur because events can overlap in time, and noise or loss of signal can obliterate a significant part of the signature. Archival data is plentiful, but assigning labels to this data is an expensive, manual process.
The other application, Guidance, Navigation, and Control (GNC), distinguishes normal from abnormal signatures in order to help controllers decide whether the onboard guidance components are functioning normally. Events last ten minutes or more. Actual (as opposed to simulated) failures are, fortunately, extremely rare, but because of the paucity of data, defining the appropriate level of generalization from sparse training data and estimating the confidence in the classifier are difficult.

Event Detection.

Most of the time the continuous EGIL data stream contains only noise, indicating steady-state loads on the onboard devices. By experimentation, we learned that we could identify most device activations by differentiating the data stream and thresholding the result. This method usually flags events in such a way that the signatures appear at a predictable offset in the time window; thus the pattern recognition modules do not need to resolve translational ambiguities. Another kind of translational ambiguity is removed by subtracting an average initial load value from the samples passed to the pattern recognition modules. The pattern recognizers, therefore, see only the load associated with the device that triggers the event, without the quiescent (DC) load due to other devices on the same bus. One other critical piece of information extracted by the event detector is which of the three phases on the electrical bus are active. This information separates the signature classes into single-phase and multi-phase classes, making subsequent discrimination easier.

Data Management.

When managing our training data became a major headache, we built a DW tool using an off-the-shelf indexed-file component (GDBM) and an interpretive X-Windows-based graphical interface (TCL/TK). The DW runs on Unix workstations, supports data visualization, classification, and formatting, and is soon to be extended for use with post-flight analysis.

System Architecture.

The two applications are running on several flavors of Unix workstations and interact with the controllers by means of an X Windows/Motif interface. All original code is written in C. Whereas quite a few software modules are applicable to more than one application, they may be used in different contexts. For example, filters to remove bursty noise spikes prior to processing the data stream are used in both the EGIL and GNC applications, but they are not invoked by the same modules nor are they invoked in quite the same way. In order to reuse such modules in multiple applications, we developed an efficient "plug-in" interface to replace hard-coded connections between modules.
Each module (data acquisition, spike filter, FFT, event detector, etc.) is written to conform to a plug-in interface. Plug-in services include initialization, termination, data distribution, and timing. When a module is provided with data via the data distribution interface, it operates on that

data and then can request that the plug-in controller pass output products to the module's recipients. The connections between processors and recipients are made separately from the modules in a dataflow module. The dataflow modules are presently hand-coded in C; future versions of the toolbox, however, will provide the ability to graphically select and connect modules.

Pattern Recognition.

We have experimented with a variety of pattern-recognition algorithms in order to build a library of PR modules. The NETS package (developed by the Software Technology Branch at JSC [1] has been successful for building feed-forward neural network classifiers. *Ad hoc* network architectures have also been used with success, notably a basis-function network combined with principle-components projection that strongly localizes the set of active function nodes [4]. Our experiences, positive and negative, with network classifiers are in concurrence with those documented by others, e.g., [3].

We have also implemented a more conventional statistical classifier that first extracts features from the events and then applies a Bayesian discriminant calculated from these feature values. Since feature extraction is usually a tricky, manual process, we worried about how feature-based classifiers might be used in an automated environment. In response we developed a method for automating the feature-extraction process based on a genetic algorithm. The features constructed by the algorithm can be used with any classifier method, including networks and decision trees [2]. With the addition of Fourier and wavelet transforms, nearest-neighbor and local-linear models, our repository of pattern classification techniques is growing rapidly.

User Interface and Configuration Builders.

Currently each application interacts with the users via an X-Windows/Motif interface. Work remains to be done on a user-definable interface builder tool and a system configuration tool, but a consensus is developing on what such an interface should include. For example, the Mission-Control venue requires that the flight controllers have a very high confidence in the correctness of the application's outputs. The user interface bolsters this confidence by making available on the display both the signature waveform and the system classifications. Controllers can, therefore, correct an occasional incorrect diagnosis and at the same time develop confidence in the accuracy of the system.

SUMMARY AND FUTURE PLANS

The results of our work to date on the Automated Signature Recognition Toolkit present a number of avenues for future work. One important direction is to continue development of specific user applications which contain the core pattern recognition tool set. As designed, multiple end-user applications should be easily created from a common system architecture, revolving around plug-in pattern recognition modules. Each end-user application will utilize pattern recognition techniques tailored to the signals or patterns for that particular console domain. New console areas will be added on a regular basis until all Mission Control Center positions with relevant data have been evaluated.

Another important direction for this work is to provide a well defined, categorized database of patterns for evaluation and testing of various algorithms. In the process of preparing the existing tools and evaluating their performance during Shuttle missions, we have gathered and classified a large amount of real-world data that is available offline for testing and comparing classification algorithms.

Finally, future challenges include the integration of expert rules with statistical pattern analysis and utilizing regularities in the temporal sequence of signature events.

References

[1] Paul T. Baffes, Robert O. Shelton, and Todd A. Phillips. NETS user's guide (version 3.0). Technical Report JSC-23366, Lyndon B. Johnson Space Center, Houston, TX, 1991.

[2] Philip Laird and Ronald Saul. Automated feature extraction for supervised learning. In *Proceedings of the IEEE Conference on Evolutionary Computation*, New York, 1994. IEEE Press.

[3] Justin D. Paola and Robert A. Schowengerdt. A review and analysis of neural networks for classification of remotely sensed multispectral imagery. Technical Report 93.05, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, 1993.

[4] Robert O. Shelton. Pattern recognition tool kit for strip chart signature analysis. In *Proceedings of the First Department of Energy workshop on Applications of Neural Networks in Materials Science*, 1994.