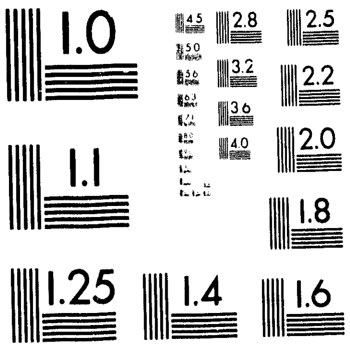
1.0
4.5
5.0
5.6
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

2

Conf-9310131--3

UCRL-JC-112639
PREPRINT

Automated Analysis Tools for Reducing Spacecraft Telemetry Data

T. J. Voss

This paper was prepared for submittal to the
1993 International Telemetering Conference
Las Vegas, Nevada
October 25-28, 1993

April 26, 1993

Lawrence Livermore National Laboratory

970

AUTOMATED ANALYSIS TOOLS FOR REDUCING SPACECRAFT TELEMETRY DATA

T.J. Voss
Lawrence Livermore National Laboratory

Keywords

telemetry analysis, data reduction, software toolkit

Abstract

A practical description is presented of the methods used to reduce spacecraft telemetry data using a hierarchial toolkit of software programs developed for a UNIX environment.

Introduction

A project requiring the design, implementation and test flight of small, lightweight spacecraft was recently conducted. This spacecraft development required hundreds of tests and integrations of subsystems on several special purpose testbeds, with each test creating large amounts of telemetered data. This paper focuses on the automated analysis and reduction of data which is telemetered from one of the key subsystems, the Probe. The telemetry system for the Probe is described in [1] and in [2]. A typical telemetry stream from a testbed run averaged 50 Megabytes of raw data, containing over 1600 system variables. The large telemetry file (raw data) sent from the Probe was decoded and decomposed into a large number of smaller Break Out Files (BOFs) containing variables with timestamps, and image files.

Motivations and Current Practice

Key motivations for automating the analysis process came from the requirements for reducing large amounts of data in short time periods into small digestible chunks which could be used by engineers and scientists to make observations of the subsystems for which they had responsibility. Without the aid of these automated tools, the predominant methods of performing data analysis could be described as repetitive, manual, visual inspection of two-dimensional data plots, followed by heuristic evaluations. A variety of tools (computer programs) and steps were used by the individuals performing the analyses. It was common for an analyst/engineer/scientist to receive complete telemetry dumps from half-a-dozen or more experimental "runs" in a single day. This meant that each person was repeating the same data reduction techniques while trying to correlate experimental changes found in the data to changes in, or attributes of, their respective subsystem. Generally, an individual could not inspect all the experimental runs.

What was needed was a systematic yet flexible, reusable, set of tools that could be quickly customized to the needs of the individual analyst in order to reduce the enormous amount of data into useful information. Additionally, a set of tools that could grow in complexity to meet the changing needs of the experiment was sought.

Design Approach to Automating Analysis

These steps outline the process of automating the data analysis:

1. Critical subsystems were selected as candidates for automated analysis.
2. Existing analytical practice as it applied to the selected subsystem was studied.
3. Whereever heuristic methods were used they were transformed into numeiical methods.
4. Numerical methods requ.red for any analysis were organized into a hierarchy of tools by taking advantage of the similarities of different data analysis processes, and using a top-down approach.
5. A common format was selected for data storage and interchange.
6. A set of primitives were created that could perform basic numerical and logical functions, data reductions or comparisons.
7. A set of intermediate level utilities were created that utilized the primitives to accomplish larger tasks.
8. A set of high level, specialized routines were developed that could organize and report useful information, again using the lower level codes.

Implementation

An example case is presented which follows this methodology.

The Probe Attitude Control System (ACS) was selected as a candidate for telemetry data analysis automation.

Existing practice was described and documented by one of the controls engineers working on the ACS.

Here are the steps taken to automate a portion of the ACS analysis.

The engineer's heuristic method was stated as, "check for consistency between inertial position information [measured gyro data] and calculated/estimated position information."

This was transformed into numerical methods as:

i) compute the difference between estimated and inertial (gyro) values for every available time point for any given run. The existing telemetry stream contained:

- estOMEGA(x,y,z) == estimated positions for each of 3 axes
- gyro(x,y,z) == measured gyro positions for each of 3 axes
- gyroBIAS(x,y,z) == measured/dynamic gyro biases for each of 3 axes

Thus:

$$estVSgyroDIFF(x) = estOMEGA(x) - 20(gyro(x) - gyroBIAS(x)) \qquad \text{(EQ 1)}$$

$$estVSgyroDIFF(y) = estOMEGA(y) - 20(gyro(y) - gyroBIAS(y)) \qquad \text{(EQ 2)}$$

$$estVSgyroDIFF(z) = estOMEGA(z) - 20(gyro(z) - gyroBIAS(z)) \qquad \text{(EQ 3)}$$

[Note: scalar multiply by 20 was needed for units conversion since estimated values and inertial (gyro) values were scaled differently by the reporting subsystems].

ii) find the mean and standard deviation of this difference over the time-sampled interval; where mean is given by

$$\mu = \frac{\sum_{i=1}^{n} x(i)}{n}$$

, and the standard deviation is given by

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (\mu - x(i))^2}{n}}$$

iii) Define a threshold of acceptable difference, such as a small factor times the standard deviation: $t = k\sigma$, where $k \in [1, 2, 3]$.

iv) Check and report all points in time where the variable of interest has a distance from the mean which is greater than the acceptable threshold: if $(|\mu - x(i)| \geq t)$ then report x(i) and timestamp(i).

The points reported by this detection then will represent those instances where the estimated position and the measured inertial position differ by a significant amount, and deserve closer scrutiny. Further analysis could entail taking the time values of the points above, x(i) and timestamp(i), and using the *tsift_pl* primitive to extract other variables of importance around an interval denoted by each timestamp(i) value, and then using *paste_pl* or *vmerge_pl* primitives to assemble these variables followed by invoking a graphics plotting tool for visual examination, or other cross-correlation tools.

Common Data File Storage Format

A common format for data storage was defined. The dot-pl file used by a local popular interactive 2-D graphics plotting and analysis tool, *px11*, was chosen because of its simple human-readable, plain ASCII text input format. (The project widely used UNIX platforms). Additionally, this allowed all the intermediate results to be readily input to either *px11* or to a popular commercial data analysis package, IDL/TADA, for graphing and visual inspection.

A dot-pl file has a simple header of a few lines (all denoted by a leading poundsign "#" or single quote " ' ") containing descriptive comments, title, list of variables contained in the file by column position, and an end-of-header marker. This header is then followed by the columns of data as defined in the header variable list, each column separated by whitespace, and each record of data delimited by a *NEWLINE* character. Common practice was to record timestamps in the first column followed by a family of related variables sampled at that time. Any number and mix of variables might be present. The variables found in any particular dot-pl file were determined by the telemetry decoder which dissected the composite telemetry stream into telemetry Break-Out Files (BOFs).

In order to allow complete flexibility in analysis it was necessary to allow decomposition of these BOFs into separate files each containing a single variable and its corresponding timestamps; hence a *two-col-*

umn dot-pl file became the common file format upon which all of the automatic analysis tools operate; where the first column gives the timestamp and the second column gives the variable value at that time, (a consistent Mission Time was used throughout the Probe flight software).

Primitive routines

A set of primitives were created to perform basic file manipulations, numerical and logical operations, i.e. logical AND of conditions (logical OR can be done by simply concatenating or pasting the desired sub-sets of data together). Additional primitives were created to perform more specialized reductions; i.e. compute mean & standard deviation and threshold detection, compute duty cycle and peak value, detect and count zero-crossings, compute linear best fit by least squares, etc. (See Table 1).

TABLE 1. Primitive Analysis Routines

routine name	function performed
getcols_pl	get a variable or list of variables
getvars_pl	get a variable or list of variables
paste_pl	concatenate together two sets of variables
vmerge_pl	merge together two sets of variables
add_pl	scalar add to dependant variable
sub_pl	scalar subtract from dependant variable
scale_pl	scalar multiply dependant variable
vdiff_pl	vector difference two dependant variables
shift_pl	(time) shift variables
interp_pl	interpolate variables
subtime_pl	scalar subtract from time (independant) variable
tsift_pl	filter (logical AND) variables
trimMP_pl	trim variables to Mission Phase (time interval)
analyze_pl	compute mean, standard deviation, perform threshold detection
dutycycle_pl	compute duty cycle and peak value ratio
leastsqrs_pl	compute linear best fit by least-squares
zerocrossings_pl	detect and count zero crossings

Intermediate Routines

Once the data format and primitives were created then a set of intermediate level utilities were implemented which utilized the primitives to accomplish the larger task of spacecraft ACS analysis.

In this example case, with the objective of checking the consistency between inertial and computed position, routines were created to perform each of ten necessary comparisons for the ACS analysis. These intermediate level codes (See Table 2) used several primitives to do their computations and analysis.

TABLE 2. Intermediate Routines for ACS Analysis

routine name	function performed
ck_est_vs_gyro.{x,y,z}	analyzes estimated vs. measured X,Y,Z-axis position
ck_est_vs_pred_q{1,2,3,4}	analyzes estimated vs. predicted Quaternion[a] 1,2,3,4
ck_est_vs_pred_w.{x,y,z}	analyze estimated vs. predicted X,Y,Z-axis position

a. Quaternions are position information often used in spacecraft control systems as they reduce the amount of coordinate computations necessary for attitude control.

High-Level Routines

Finally, specialized high-level routines were created to perform a summary reduction and report the analysis results in a single concise format, which for this example case was done by routine *acsEstimatorConsistency.*

Additionally, the routine *acsEstimatorConsistency* can be invoked as a subordinate process from routine *run_acs*, which can be given a list of "runs" to batch process, allowing the unattended examination and reduction of a series of runs from a single invocation.

A run summary report is shown in Table 3., which indicates the count of the occurrences (num_GTL)

TABLE 3. Summary Report of ACS Estimator Consistency

# Probe Telemetry Report
# ACS State Estimator Consistency
# Run Identifier: r10.06.92e
# This report generated on 10/07/92;08:57:15 by tjv

Nvals	Mean	Std_Dev	num_GTL	test
813	-0.000102	0.001317	6	est_w.x vs. gyro_data.x
813	0.000021	0.000387	9	est_w.y vs. gyro_data.y
813	-0.000011	0.000419	7	est_w.z vs. gyro_data.z
813	0.000007	0.000086	7	est_q1 vs. ncycle_pred_q1
813	0.000002	0.000018	4	est_q2 vs. ncycle_pred_q2
813	0.000010	0.000052	6	est_q3 vs. ncycle_pred_q3
813	0.000000	0.000001	8	est_q4 vs. ncycle_pred_q4
813	0.000229	0.002454	6	est_w.x vs. ncycle_pred_w.x
813	-0.000062	0.000720	7	est_w.y vs. ncycle_pred_w.y
813	0.000013	0.000809	7	est_w.z vs. ncycle_pred_w.z

when the mean difference between expected and computed position was greater than allowed. Detailed reports are also available as an option which itemize the points by timestamp to reveal when (in Mission Time) the anomalies occurred This report gives the controls engineer a quick look at ten different computations showing the length of the experimental run (Nvals), the mean difference between estimated and measured parameters (Mean), the standard deviation of that difference for that run (Std_Dev), the number of points which exceed a selected threshold difference (num_GTL), and the test/variable names (test).

Following this quick-look, the analyst also could organize these run reports and perform various statistical operations on them; for example, catalog all the experimental runs by number of differences exceeding a threshold, as done by utility, *rankAcsByErrors* and shown in Table 4.; thus giving an idea of

TABLE 4. Top Ten Runs by num_GTL based on gyro_data.x vs. est_w.x

rank	run_id	nVals	Mean	Std_Dev	num_GTL
1	r08.18.92b	1097	0.000006	0.000006	0
2	r08.13.92i	327	0.000212	0.000672	2
3	r09.22.92a:	817	-0.000015	0.000810	2
4	r08.13.92h	327	-0.002558	0.004322	3
5	r10.06.92a	885	0.000091	0.001037	4
6	r10.06.92d	821	0.000009	0.001220	4
7	r08.13.92d	592	-0.000157	0.001550	5
8	r08.13.92f:	779	-0.000212	0.001379	5
9	r08.18.92i	3323	-0.000013	0.000748	5
10	r08.13.92b	674	-0.000022	0.001575	6

experimental progress or regression. While this example is straight-forward, the methodology and set of primitives created allowed for many varied and complex analyses. For example, variables could be studied over a specific mission time phase, and/or correlated with combinations of other variables or functions, by using other primitives.

Profile Comparisons and Adaptive Analysis

A data "profile" (example waveform with a specified error margin) could be taken from previous experimental data, and then used as the basis of comparison for future runs - updating the profile as needed to refine the system under study. If this feedback process were used consistently it would form the basis of a simple *adaptive* automated analysis process.

This also allows the analysis to be uniquely customized to the system under test (allowing arbitrary waveforms) rather than requiring comparisons be confined to computed static numerical functions.

Performance of Automatic ACS Analysis

The quick-look automated analysis summarizes and pin-points areas which require more detailed analysis, while helping to insure that problem areas do not go undetected, since *all* the experimental data can now be examined for every run in a short time period.

A typical ACS analysis took under 3 minutes on a Sun SPARCstation IPC, with the files remotely mounted from a Sun-4 server over an Ethernet 10Mbps LAN; and under 2 minutes on a Sun-4/490 with locally mounted files; thus performing the examination of that batch of half-a-dozen experimental runs per day in under 12 minutes!

Conclusion

The development of a re-usable, flexible toolkit to perform data reduction and analysis of telemetry streams from experimental runs helps shorten and simplify the design and implementation of complex systems. The forte of automatic analysis tools is reducing amounts of data that humans would find overwhelming and may tend to gloss over, into consistently analyzed informative reports, which allows the analyst/engineer/scientist the ability to rank outcomes of experiments and measure progress.

The approach used provided a hierarchial structure, which allowed for flexible expansion, while minimizing the amount of effort required to build the toolkit. The complete set of primitives were completed in a period of two weeks, followed by intermediate and higher- level routines which often were completed in less than an hour - this was the result of two beneficial factors: (1) the higher-level routines had the advantage of calling primitives to organize and perform tasks, and (2) the UNIX file system and shell environment provided an easy inter-connection and easy authoring of programs.

Many data analysis packages excel at either batch processing or interactive operation. Experimental science often demands both - a good interactive tool (typically graphical) which permits the user to visualize and analyze-as-they-go, and a good batch processing capability (typically mathematical & statistical) to allow the consistent reduction of data to be performed automatically once the methods have been developed; which implies the need for a good language which permits the user to express their analytical desires and incorporate them into automatic reduction methods worthy of reuse.

The toolkit discussed contains a few of the many analysis routines one might need to form a broad-based kit, and it performs its functions in batch processing style resulting in tabular statistical numeric reports - requiring that the user invoke other routines to provide interactive graphical interfaces. Two things directly impact the interconnection of routines: (1) sharing a common data format for information interchange; and (2) a method of invoking a peer or subordinate routine, which will operate on any data given to it; In this toolkit the author chose the ASCII dot-pl file format as the common data format, and the UNIX C-shell language, with its inherent pipes and redirectors, argument passing, et.al. capabilities, as the means of invoking subprocesses able to operate on data passed to them.

All of the utilities and their manual pages are available as public domain sources. The utilities are written in the C programming language and in the UNIX C-shell language. The dot-pl file header processing is a modular code piece which can be readily modified to accommodate different storage file formats, thus requiring minimal changes to the main portions of the codes for those needing to analyze data stored in other formats.

Work performed under the auspices of the U.S. DOE by LLNL under contract no. W-7405-Eng-48.