Automated Application of Calibration Factors on Telemetered Data

J. R. Kalibjian, T.J. Voss, J.J. Yio
Lawrence Livermore National Laboratory

Keywords
calibration factors, telemetry post processing

Abstract

A long standing problem in telemetry post processing is the application of correct calibration factors to telemetered data generated on a system which has had a history of hardware changes. These calibration problems become most exacerbated when old test data is being examined and there is uncertainty as to hardware configuration at the time of the test. In this paper a mechanism for introducing a high degree of reliability in the application of calibration factors is described in an implementation done for Brilliant Pebbles Flight Experiment Three (FE-3).

Introduction

In order to obtain high reliability in the application of calibration factors, the flight software producing telemetry data must be tightly bound to the physical hardware configuration of the spacecraft. In tandem with this, there must be strict revision control on the flight software and the file maintaining the spacecraft calibration factors, where particular attention is paid to changes in the calibration file necessitated by spacecraft hardware modification. Utilities may be developed to select calibration data that will be applied to uncalibrated telemetered data corresponding to a particular hardware configuration of interest. These utilities may be accessed by high level data analysis tools to allow for user transparent calibration of flight or test data.

Flight Software Revision History

In order to bind flight software to a hardware configuration, a new version of the flight software must be compiled after a hardware modification has been made. This is not unreasonable to assume if the hardware which has been modified is used by the on board software in a decision making process (i.e. the calibrations in the flight software need to be modified). The requirement becomes a nuisance if the hardware that is modified is merely "blindly" sampled and telemetered. After compilation and linking,

a utility must be run to insert the day, date, and revision number of the flight software into a designated "blank" area of the executable image. When the flight software is operating, there will be a requirement to telemeter the contents of this previously blank location several times during the mission to insure the date of compilation will be available in the decoded telemetry. After this information is broken out of the telemetry it can become available to utilities examining the data.

Establishing Rules for Revision History of Calibration Files

Insuring that the correct calibration data is used when performing a calibration, requires that a correspondence exist between the data to be calibrated and the calibration factors to be applied to the data. That correspondence may be either a one-to-one or a one-to-many relationship; i.e. one set of calibration factors being the correct set to be applied to one or many sets of experimental data. To determine the correct match, one must know the relationships between all pairs of data and calibration factors. Often these relationships are known only to the individuals who have made a specific change in the experimental hardware or software. These changes may have also necessitated changes to the calibration files. If more than one person makes these changes, and if many sets of data are generated, then the number of relationships which need to be remembered becomes large and therefore the probability of correctly matching the data and its calibration information becomes small. An automated mechanism which can insure the correct match would greatly increase the confidence of data analysts when viewing correctly calibrated data. Figure 1 depicts this process.

The calibration mechanism described here requires:

> (1) any software which generates experimental data to be calibrated must "stamp" the software compilation date into the data set;
> (2) the run date, which gives the exact time of the experimental run, must also be included in the data set;
> (3) coordinated revision control of the software and the calibration data and its correlation to hardware changes are required.

Items (1) and (2) can be fully automated by software given a wall-clock, while item (3) requires that persons performing the revisions adhere to a standard procedure described below.

The automatic calibration software *calib* (a UNIX utility) will use the information implied from requirements (1) and (2) to determine the correct match between data and calibration factors. This is achieved by searching for the most recent version of

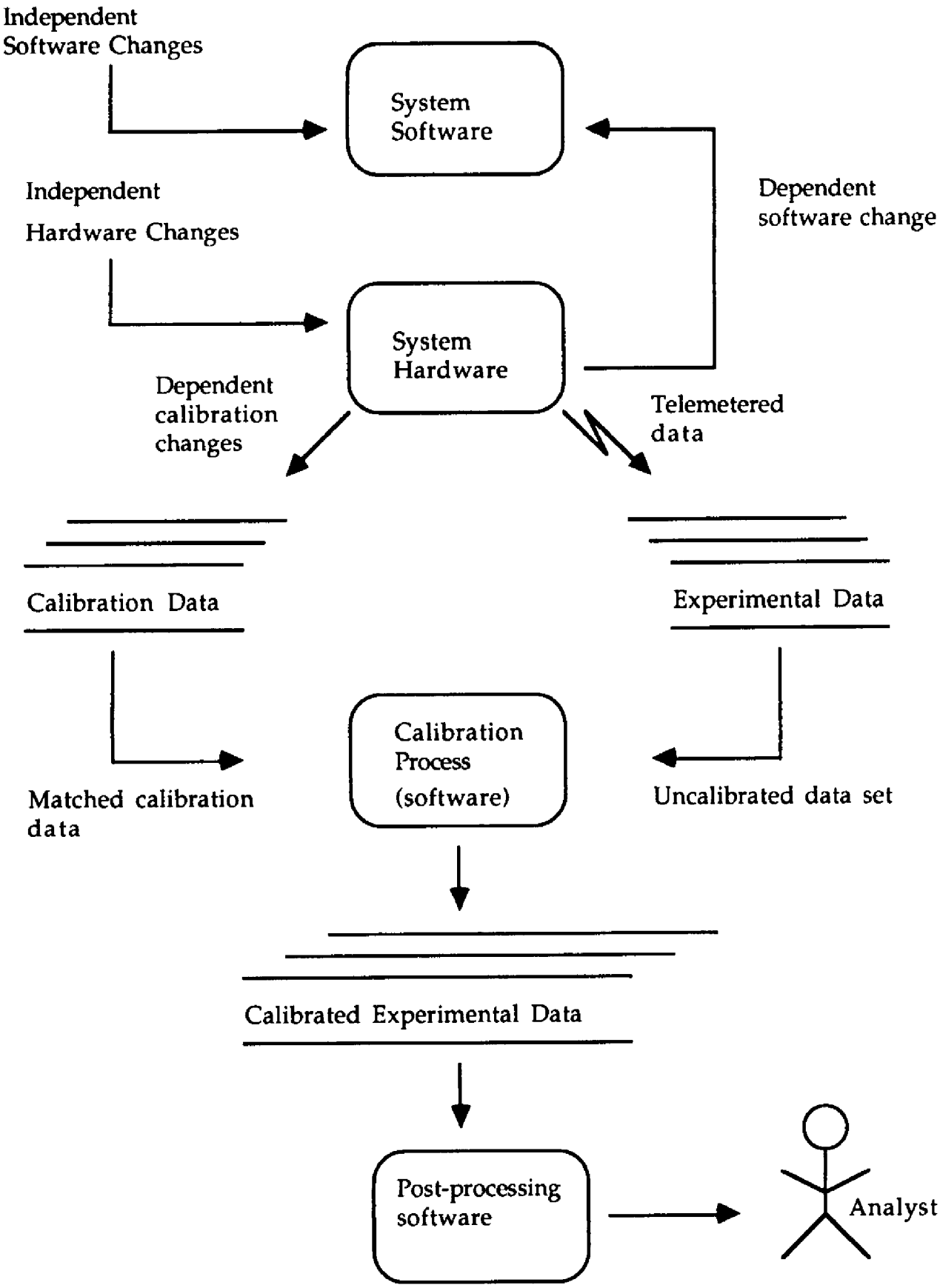

Figure 1. The Calibration process.

calibration factors that existed in a controlled revision tree equal or prior to the rundate or the compile date, whichever applies to the datum being calibrated. If the datum being calibrated is a software only variable (i.e. hardware changes do not effect this variable), then the compile date should be used; otherwise, the rundate should be used. This is because a hardware change could have occurred after the software was compiled and before the run took place, or a hardware change could have occurred which did not require any changes to the software, and thus a re-compile was not performed. It is recommended that all project calibration data be stored in Revision Control System (RCS) files, according to a standard procedure which is described in this paper. RCS is a set of UNIX utilities which manages multiple revisions of text files and is commonly available on many UNIX systems. It is also available from the Free Software Foundation.

Generally, calibration data files will need to be updated when one of the conditions occurs:

(1) changes in system hardware;
(2) changes in system software or error corrections;
(3) back-annotations or post-modifications to calibration data;

Thus, there will be multiple revisions of calibration data within the RCS file that correspond to the earlier status of the experimental system (hardware and software). In order to successfully use the calibration data, one must first determine which revision of the data is desired, and then "extract" the desired revision from the RCS file. This could be a difficult job depending on the information available to an analyst concerning changes and dates of experimental runs.

Procedures for Maintaining a Calibration File Under Revision Control

All calibration data must be maintained under RCS and this is accomplished by checking-in the file(s) containing the calibration factor data using RCS's *ci* (check-in) utility.

A RCS file may be represented as a tree structure which has a trunk, branches and leaves. RCS is aware of the ancestral relationship between any two revisions (nodes) in the tree; however, the chronology of the files is merely recorded as an attribute, i.e. the tree growth is not necessarily ordered by time; rather, by relationships among the revisions. Further, abuse or ignorance and mis-use of RCS utilities can result in a RCS tree with an awkward structure and even erroneous dates or other attributes. Thus, adherence to this standard method is recommended .

Each time a hardware change occurs, a new major revision (new RCS branch) should be created. This process is performed below on a UNIX system:

```
% ci -rX.Y EXPERIMENT.cal
```

Here, X.Y is the new rev ID, and note that X=(previous X) + 1 and Y=0, for each hardware change. Each time a software change or error correction occurs, a new minor revision (RCS branch revision) should be created. This is shown below

```
% ci -rX.Y EXPERIMENT.cal
```

Here, again, X.Y is the rev ID, and note that X = (previous X), and Y= (previous Y) + 1 for a minor revision.

Hence, a typical RCS tree would have this generic structure:

RCS revID	Reason for Change Code H=hardware E=software/error correction
1.0	Initial File
1.1	E
1.2	E
2.0	H
2.1	E
3.0	H
3.1	E
3.2	E
3.3	E

Notice that normal revisions due to hardware changes or software changes/error corrections produce a RCS tree with rev IDs that have only two (2) decimal fields; e.g. 1.2.

If it becomes necessary to patch or fix old versions already in the RCS file, then RCS branches with leaves will be used, which have a RCS revID with four (4) decimal fields, e.g. 1.2.1.1. The following example assumes a patch to a file with revision number X.Y:

```
% co -1 X.Y EXPERIMENT.cal          # Check-out rev to be fixed

% vi EXPERIMENT.cal                 # Use text editor to make
                                    # change
```

```
% ci -u -rX.Y.1 EXPERIMENT.cal     # check-in corrected ver
```

Subsequent patches to X.Y branch would be carried out as shown below,

```
% co -lX .Y.1 EXPERIMENT.cal       # check-out most-recent leaf
                                   # in branch X.Y.1

% vi EXPERIMENT.cal                # Use text editor to make
                                   # change

% ci -rX.Y.1 EXPERIMENT.cal        # check-in corrected version
                                   # as new leaf in same branch
```

Note that the check-in rev ID is "X.Y.1" (no matter how many times this branch has been patched) so that RCS will check-in this patched revision with the next numbered sequence at this branch starting with X.Y.1.1.

Hence, the aforementioned RCS tree with hardware changes, software changes/error corrections and patches, would have this generic structure:

RCS revID	Reason for Change Code H=hardware E=software/error correction P=Patch
1.0	Initial File
1.1	E
1.2	E
1.2.1.1	P (First patch to 1.2)
1.2.1.2	P (2nd patch to 1.2)
2.0	H (New branch)
2.1	E
3.0	H (New branch)
3.1	E
3.2	E
3.2.1.1	P (First patch for 3.2)
3.3	E

It should not be necessary to create more than one branch at any trunk node. Doing so would unnecessarily complicate any later extraction.

A Practical Example, the FE-3 Experiment

The calibration concepts discussed above were implemented for the third flight experiment in the Brilliant Pebbles test program (known as FE-3). The goal of the

Brilliant Pebbles Program was to demonstrate an autonomous interception and destruction of a sub-orbital thrusting target by a light weight sub-orbital spacecraft.

The FE-3 Flight Software

The FE-3 flight software had access to thirty analog devices which required calibration (excluding cameras). Of the thirty analog devices, only six were actually used by the flight software in a decision making process. The other twenty-four channels were merely sampled and telemetered. After the compilation and linking process, a utility called *dmsstamp* was employed to place time, day, date, user name, and variant (version) information at a pre-established location in the executable image. The information at this location was periodically telemetered during the the mission experiment. The telemetry decoder extracted this information during mission post processing and placed the time stamp data in a break out file. Another utility *getCompileDateNTime* extracted this information from the break out file and output the compile date and time information to standard output. Thus, this information could be piped to other utilities for their use.

FE-3 Calibration Files

The calibration file format used for the FE-3 experiment is shown below,

```
# LLNL Unit Calibration File
# $Id: probeLLNL.cal,v 1.4 92/08/06 16:14:30 kal au16 $
#
analCpuA
time                    0
qotdata                 0
samplesA  0
pumpedHydrazinePres 1 -.9214467 1884.941     "PSI (LSB=.921)"
excite1ChkVoltage   1 -.009768 20  "Volts (LSB=.00977)"
accelX    1 -.002429853 4.975 "Gs (LSB=2.43e-3)"
accelY    1 -.002429853 4.975 "Gs (LSB=2.43e-3)"
.
.
```

Calibrations were defined with respect to the packet type which contained the data variables to be calibrated. In the fragment shown above, the calibrations for data variables gotdata, samplesA, pumpedHydrazinePres, excite1ChkVoltage, accelX, and accelY are given for packet ANALOG_CPU_A. The first field references the variable to be calibrated, the second field indicates the calibration method to be used, the next fields contain the calibration factors themselves (the number of these fields is dependent on the calibration method used, e.g. linear, cubic, etc), the next field is a text string that describes the units of the variable after it has been calibrated.

Comments can be includes anywhere in the file as long as they are preceded by the "#" character. There were four calibration files maintained, each corresponding to a distinct FE-3 vehicle,

probeGTU.cal	Prototype vehicle at flight integrator facility
probeLLNL.cal	Prototype vehicle at Lawrence Livermore.
probeFLT.cal	Flight vehicle.
probeNTS.cal	Test vehicle at Nevada Test Site.

A Utility to Extract Calibration Files: *cobydate*

Cobydate is a UNIX utility to check-out RCS revisions, similar to RCS's "co" utility-----only with some special provisions for finding a revision based only on "date/time"; specifically,

> It does this by making an assumption that the user wants the most recent revision (highest numbered revision) in any branch whose main trunk node was checked-in equal or prior to date.
>
> This was done by dumping the rlog of the RCS file into a temporary file, then looking through the rlog dump at the various trunk nodes, searching for the trunk node created equal or just prior to date, and then determining if that trunk node has any branches.
>
> If the trunk has branches, then the most recent leaf node (highest numbered revision in this branch) is extracted from the RCS file; otherwise, the trunk node is extracted.
>
> The extracted revision is checked-out "unlocked".
>
> If no trunk nodes qualify, based on the date given, then no action is taken, and a message is written to standard output.

Although *cobydate* was written originally to handle FE-3 calibration data, it has a broader general applicability to any experimental data tracked by the methods described in this paper.

Cobydate was written in ANSI-C and relies upon RCS's "co" and "rlog" utilities. It also incorporates RCS's free-format data/time parser to convert user supplied dates into internal format for comparisons. It correctly handles date/time conversions and accounts for leap years, daylight savings time and time zone differences. An example

of cobydate usage is shown below. Assume the user had data he/she wished to calibrate from a run on the flight integrator's prototype unit that took place on July 5, 1992 at 10:02 a.m.

```
% cobydate -p probeGTU.cal "7/5/92 10:02 am" >probeGTU.cal.7.5.92a
```

This command line will create the file "probeGTU.cal.7.5.92a" containing the correct calibration factors for a run which took place at that time.

Stand Alone Calibration System

There are two approaches to calibration application. One is for the user to take uncalibrated data into an analysis tool and calibrate it there as a preliminary step to further analysis. The other is to calibrate with a stand-alone utility, such that data files so calibrated may then be analyzed with different analysis tools and perhaps by different users. To support the latter approach, a command-line utility named *calib* was developed. This utility calibrates one data file at a time. It relies on the extraction utility described earlier to scan a calibration file (.cal file) specified by the user, for information on whether and how to calibrate each of the variables found in the data file. The input file and the calibration file are specified on the command line as follows:

calib <input_data_file> <calibration_factors_file>

e.g.

```
%    calib     r07.05.92a.analoqCpuA.pl      probeGTU.cal.7.5.92a
```

or,

calib <input_data_file> -<hardware_type> <date> (alternate form)

e.g.

```
%    calib r07.05.92a.analoqCpuA.pl     -qtu 7/5/92
```

The first syntax form may be used independently of the checkout system, so long as the calibration factors file is of appropriate format. The alternate form can be used only if the calibration factors file is coming from the checkout system and the user wishes *calib* to call *cobydate* on his/her behalf.

The Input data file must conform to a custom format created by the FE-3 telemetry decoder *tdfe3*. In this format, like many other ASCII time-series files, data records are lines of ASCII characters; when listed, they show values for time and data variables in distinct columns across the display (separated by white space). Any data file in this format can be calibrated with this utility, provided the data records are preceded by two header records: one lists the name and data type for the time variable and all data variables, in the same order their values appear in each data record; the other marks the end of header.

Calibration inside an Analysis Tool

There is one drawback to calibrating data prior to analysis session(s). That is: a calibrated data file kept around becomes outdated, as soon as the calibration factors file gets updated with changes (correcting for typographic or sensor calibration errors) -- resulting in a new revision for the desired hardware version. For this reason, FE-3 analysts were provided with the option of having data calibrated at the time of analysis. The FE-3 data analysis tool, *TADA*, was modified so that it would call *cobydate* the way *calib* did. It would retrieve the latest revision of the calibration factors file for the hardware type and version that was used to generate the data set being analyzed. *TADA* would initiate calibration of the data accordingly, each time the user selected a set of variables to plot (which needed to be calibrated) .

Calibration in this manner may be characterized as "calibrating data as needed on the fly". It is transparent to users. As implemented in *TADA*, it incurs processing time that is hardly noticeable to users, even in the fast-paced context of a window-based user interface which *TADA* offers. One major reason accounting for this favorable performance is that *TADA's* input function is quite fast in itself. (Also, *TADA* limits the number of graphs to four in a plot). *TADA* inputs data quickly, because it presumes that someone has taken time to convert the data files (of a new data set) from the ASCII format (which is human readable) into a "variable contiguous" binary format (which is machine efficient), that it requires. This concept was developed by the Brilliant Pebble's flight integrator, Ball Aerospace Group.

Conclusion

Binding flight software compile times to hardware configurations and maintaining strict revision control of calibration factor change necessitated by hardware modification provide the links needed to produce a reliable automated calibration system. The key to the capability of the system lies in the utilities (*cobydate*) which select the appropriate calibration file in light of the flight software compile time and

apply the calibrations (*calib*). These concepts were used in the automated calibration system implemented for the Brilliant Pebbles Flight Experiment Three experiment.

Acknowledgments

The authors gratefully acknowledge the following individuals: J. Fleming, L. Fleming, D. Eva, B. Hedeline, B. Henderson, and M. Henderson.

This work was performed under the auspices of the U.S. Department of Energy by LLNL under contract number W-7405-Eng-48.