Appendix III

NASA/NAG-1-667/R/3.0/NCSU.CSC.(DFM,KCT,MAV)/Oct-87

RSDIMU Acceptance Testing System

Version 3.0

May 1987

Fault-Tolerant Software Experiment

OVERVIEW OF ACCEPTANCE SOFTWARE AND PROCEDURES

RSDIMU ACCEPTANCE TESTING SYSTEM (RSDIMU-ATS)

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

This note applies to the releases 3.0 of RSDIMU-ATS. The release complies with the version 3.2/10-Feb-87 of the RSDIMU specifications. This system is released for restricted use by sites involved in the NASA-LaRC fault-tolerant software experiment.

Institutions participating in the program:
NASA-Langley Research Center, CRA, NCSU, UVA, UCLA
(ex-participants: RTI, UIUC).

Re-distribution and use of this system for purposes other than the ones compatible with the current experiment is prohibited unless explicit permission is obtained from the NASA-Langley Research Center coordinator for this experiment (Dr. D.E. Echkardt).

The RSDIMU Acceptance Testing System (RSDIMU-ATS) was built to help test and analyse multiversion RSDIMU procedures generated as part of a NASA sponsored fault-tolerant software experiment in progress since Spring 1985. RSDIMU-ATS is intended for use in a UNIX environment and may need to be modified if UNIX-like, or non-VAX systems are used. Part of the software needs to be recompiled if used on non-VAX hardware (e.g. SUN workstations). It was tested on VAX 11/780,785, and MicroVAX II hardware under UNIX 4.2/4.3BSD, and Ultrix1.1/1.2 respectively.

The system is shipped on a 9-track magnetic tape (standard size) in tar format at 1600 bpi, as directory tree rooted in ./fts87.

```
                                fts87
                                  |
 ______________________________________________________________________
|        |       |     |          |           |          |         |
ReadMe  accept  code  data  generators*  gold  nonVAX_host**  certify  testcases*
```

  *   shipped on request only
  ** to VAX sites shipped on request only

The following notes discuss more important features of the system.

1. ReadMe is the file containing this note.

   RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

2. The core of the acceptance testing system is located in the directory "accept". It is intended to help an experimenter run, evaluate and request corrections of programs in a semiautomatic fashion. A more detailed description of the accept system is found in ReadMe notes in the accept/ and certify/ directories. The system is based on the use

of a "golden" program for adjudication of the correctness of the answers generated by the code being tested. The RSDIMU code is tested one component (version) at the time against a pre-recorded output expected from the "golden" code. Any differences from the expected answers have to be examined in detail, and corrections justified in error correction reports. Our confidence in the correctness of the "golden" code is very high, however experimenters should still be on the look-out for discrepancies indicating possible "golden" code errors. If such are discovered all on-site testing should be frozen, and the RSDIMU-ATS distribution site (NCSU) notified immediately. Similarly, if errors are discovered in the ATS harness scripts please notify NCSU. A detailed description of the acceptance procedure is located in accept/ ReadMe notes. The basic idea is to follow an iterative correction process, i.e test-correct (one or more errors at the time)-test etc. All communication with programmers should follow guidelines given in certify/ReadMe notes. Testing involves location of the fault(s) causing the first 20 failures and its (their) removal. This repeats until no errors are detected by the supplied acceptance test set.
The same test data set is used on all programs at all sites.

An extended analysis system providing MCF profile (intensity) analysis is available on request, but it should not be used to perform acceptance testing as part of the current experiment (see note 4).

3. All the testcases in this release of the system comply with the specification version 3.2/10-Feb-87. A test case entry consists of an input record, and an output record. The latter contains what is believed to be the correct answer to the input record according to the 3.2 specs, and as generated/given by gold3v2.i.

   A set of test cases was designed and generated for acceptance testing rsdimu code. It consists of a group of 796 extremal/special value (ESV) test cases and a group of 400 random test cases. All test cases are located in the directory data/.

   A successful pass through all the test cases gives an estimated lower limit on the reliability of the rsdimu code of about 0.992 (valid for the employed sampling profiles).

   More details about the test cases can be found in the ReadMe notes in the data/ directory.

   Directory "generators" contains files and code one may need to generate the ESV and random data from scratch. It is not expected that a user working in a VAX-UNIX environment would have to do that (however, see notes in the nonVAX_host directory). In fact, it is not recommended that sites generate (or re-generate) data without consultation since rounding and other subtle differences might appear between the newly generated cases and the ones on this distribution tape, resulting in the use of somewhat different cases by different sites.

   Directory testcases/ contains raw esv test cases (text form).

4. Directory "code" is assumed to contain the rsdimu code that is being tested. If you do not wish to keep your code in that directory either use

pointers/links to the code, or change the appropriate portions of the acceptance system.

Note that RSDIMU-ATS is sensitive to the overall file structure that is used in the system, and any changes should be made only after consultation with NCSU.

It is extremely important for the success of the experiment that you keep not only the final, corrected version of each program, but that each intermediate version submitted for acceptance testing is saved and tagged with an appropriate version number and information about the changes/corrections (using the provided change form, in certify).

It is important that you promptly review all the returned error-reports. In this experiment we are not implementing back-to-back version testing and a formal automatic correlation search-and-remove loop (we have the tools, but since this was not part of the original experimental design we do not want to change the rules now). However, all the discrepancies between individually submitted and tested components and the "golden" answers should be scrutinized with extreme care in case the difference is due to the "golden" code, rather than your own code (a posible cause could be use of RSDIMU-ATS in an environment not 100% compatible with the ones in which the system was tested, see notes in nonVAX_host/ directory).

5. The "certify" directory contains code and files that are sent to each maintenance/certification team. It contains a basic rsdimu driver (to avoid interface problems) and instructions on its use. It also contains a sample input and output, and an electronic error report file.

```
*******************************************************************
*   Local site experimenters have to change/adjust the            *
*   certify ReadMe and ReadMe_to_certify notes to reflect local   *
*   electronic mail address, and local version management         *
*   environment and approach (RCS, SCCS, or similar).             *
*                                                                 *
*   You should also make all 'fts_' files in certify 'read_only'  *
*   to prevent accidental changes in the team's testing           *
*   environment (prevent changes by making yourself the owner.    *
*******************************************************************
```

Whenever a change is made in the code it is expected that the programmer(s) will record it using this report. A new version of the code, and error and change report(s), have to be returned to the experimenters together.

It is essential that each program be given a version number and associated with it the date of it creation. Every time a program is corrected its version changes and should be recorded in the correction report, as comments in the code itself, and should reflect in the file name for the new code (as kept in the "code" directory, and in the "newcode" directory in accept/).

All versions of the code, i.e. all submitted corrected code, should be preserved for future analysis. Examples are given in the "certify" directory and in the "accept" (detailed procedure). Use magnetic tapes.

If (disk or tape) space is a problem, we suggest that you store only the difference between the original version and successive corrections using for example Unix diff processor.

FOR ANY INFORMATION REGARDING THIS SYSTEM PLEASE CONTACT:

M.A. Vouk
North Carolina State University
Department of Computer Science, Box 8206
Raleigh, NC 27695-8206

Tel: 919-737-7886 (office)
919-737-2858 (departmental office)

USENET: mcnc!ece-csc!vouk
ARPA: vouk@ece-csc.ncsu.edu

Fault-Tolerant Software Experiment

ACCEPTANCE ENVIRONMENT

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

This is the basic acceptance environment. Most of the shell scripts and programs have either a help which describes its activation parameters (invoke the script without any parameters), or internal documentation. Where needed program source code is provided.

Current version of the system is intended for UNIX csh environment under either 4.2/4.3BSD, or Ultrix 1.1/1.2, running on VAX hardware.

Comparison of the values computed by programs with those using golden code is done using relative tolerance. It is possible to switch to absolute tolerances if that is desired. Do not do that for acceptance testing.

Testing tolerances are set to the following values within fts_accept and can be changed by modifying statements within fts_accept (see fts_accept help):

```
DiffBestEst = 0.00024414
DiffLinOut  = 0.00024414
DiffOffset  = 0.00024414
tolerance_mode = relative
```

Please do not use different tolerances for your acceptance testing before getting concurrence from all the other testing sites. Otherwise we shall each end up testing and correcting different things.

There is no tolerance regarding the display values (five digits), but one could allow for a difference in the last displayed digit. To avoid display related warnings and "failures" of the type: gold 4.9999 vs. computed 5.0000, and to therefore test only the display algorithm, we inject (using voteestimates) golden values for bestest and other real-valued variables prior to display computations.

The acceptance harness tests for agreement on eleven output variables (infact 59, if elements of arrays are counted separately, number of elements is given in parentheses). They are:

LINOFFSET, LINNOISE, LINOUT, LINFAILOUT, SYSSTATUS, BESTEST, CHANEST, CHANFACE, DISMODE, DISUPPER and DISLOWER.

Critical variables are: (3)BESTEST, (8)LINFAILOUT, ((1)SYSSTATUS)

Non-critical: (1)DISMODE, (3)DISUPPER, (3)DISLOWER, (12)CHANEST, (4)CHANFACE

Intermediate: (SYSSTATUS), (8)LINOFFSET, (8)LINNOISE, (8)LINOUT

All variables are checked for each test case.

For more details on the checking of variables and tolerance used see the listings of the fts_harness files, and the April 86 NCSU Working Notes from the Langley meeting (NASA.FTS/NCSU/WN/1/Apr-86), and UCLA notes from the same meeting.

To avoid accidental correctness problems output variables are "trashed" before each test case is run.

The trash values injected in the various output variables of rsdimu are as follows :

```
LINOFFSET : -9999.0
LINOUT    : 999999.0

BESTEST.ACCELERATION [1..3] : 9999999.0;
CHANEST [1..4].ACCELERATION [1..3] : 9999999.0;

DISMODE : 65534
DISUPPER [1..3] : 65534
DISLOWER [1..3] : 65534
```

The values for real variables (first four listed above) cannot occur for the current set of input data, and are highly unlikely otherwise. The display values are supposed to turn on only the G segment for the least significant digit. Boolean output variables, and user defined are initilized by the compiler (to zero).

Primary scripts:

```
fts_certify      - shell script which activates fts_accept with all.dat
                   test data and produces a correction request report
                   and test cases for the cerification team by
                   running fts_correq. Certain program naming conventions
                   and running options are built-in.

fts_accept       - shell script for constructing, compiling and running
                   harness+rsdimu code.

fts_correq       - correction request generation shell script, generates
                   a report/request suitable for mailing to the
                   maintenance teams.
```

Utility scripts and programs:

```
fts_listdata     - script for listing test cases from the test data files.

fts_prnt         - data listing program.

fts_prterr       - program produces test cases suitable for use by
                   fts_driver.p code.

fts_ter1         - block coverage computation script.

fts_lc           - lower-case filter.
```

```
fts_uc              - upper-case filter.

fts_nc              - comment-delimiters lex-based filter.


Source code and script parts:

fts_io              - sed control code to flag "integer","real",and rsdimu i/o.

fts_dbxbug          - dbx bug control code.

fts_dbxinit         - dbx initialization code.

fts_harness.declare - test harness declarations.

fts_harness.rest     - test harness body.

fts_msgtext         - correction request message.

fts_prnt.p          - source code for fts_prnt.

fts_prterr.p        - source code for fts_prterr


Documentation and examples:

ReadMe               - general information about the "accept" directory.

ReadMe_to_certify - certification procedure using fts_certify

ReadMe_accept        - using fts_accept.

example/            - directory with example outputs from an fts_accept run
                      (ncsuD7.i tested by executing:

                      fts_accept ncsuD7.i ncd7 all -c -x > test.ncd7all&
                      fts_correq ncd7 all > correq.ncd7
                      ).

newcode/            - empty directory for testing results (reminder),


Data links:

all.dat             - symbolic link to esv+random acceptance test cases.

esv.dat             - link to extremal and special value (esv) test cases.

randNCSU.dat        - link to independent random test cases (uniform profile).

randCRA.dat         - link to independent random test cases (shaped profile).

                All .dat files are in ../data.
                It is also assumed that the code to be tested is in ../code.
```

```
The fts_lc, fts_uc, fts_nc, and initial fts_io filters were written by RTI.

  The filter fts_io (integer/real, and i/o) is rather crude.
  It will miss 'real' at the begining of a line.
  It may also cause false warnings regarding use of integer and real types,
  and of i/o in the rsdimu code. In those cases hand editing and
  recompilation of <work_name>.p (rsdimu+harness) files may be
  necessary. If editing, search for two question marks ??.
  If recompiling use: pc -s -C -g -z options. Re-run fts_accept without
  the -c option.

  Note that -s compiler option yields messages regarding non-standard
  use of Pascal in the code (primarily the harness code). These
  messages should be ignored.

  Alternatively, delete the first two lines (real/integer),
  or third and fourth lines (i/o) of the fts_io code.

  fts_nc filter for comments may cause problems by making nested
  comments of type {(* comment *)} transform to {{ comment }}
  which is illegal. This filter can be excluded from the processing
  pipe in fts_accept.
```

Fault-Tolerant Software Experiment

PROCEDURE FOR ACCEPTANCE TESTING

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

```
**********************************************************************
*                                                                    *
*  For the purpose of conducting the acceptance testing (certification *
*  of the 20 programs) it is recommended that you use fts_certify.   *
*  Unless you intend to use fts_accept directly, rather than through *
*  the fts_certify facility, you do not need to read this file.      *
*                                                                    *
**********************************************************************
```

****

It is assumed that the communication between the experimenters and the maintenance personnel will be via electronic mail. It is further assumed that the experimenter has full access to maintenance personnel files, but the reverse is not true. It is also assumed that the acceptance testing is performed in csh in UNIX 4.2/4.3BSD, or Ultrix 1.1/1.2 environments running on VAX hardware (else see nonVAX_host/ directory).

I.

The testing begins by placing the code which is to be tested into the code/ directory. Enter the accept/ directory and start the initial round of testing by executing the fts_accept shell script. It is recommended that you use the -x option and benefit from the coverage information thus provided. For example:

```
     fts_accept ncsuB2.i ncb2 all -c -x > test.ncb2all&
```

NOTE: YOU CANNOT RUN TWO fts_accept JOBS FROM THE SAME DIRECTORY AT THE SAME TIME (IN BACKGROUND). THE SYSTEM WAS NOT DESIGNED FOR THAT AND YOU CAN END UP WITH A MESS. YOU CAN, HOWEVER, KEEP TWO DIRECTORIES, SAY ACCEPT1 AND ACCEPT2 AND RUN AN fts_accept FROM EACH OF THEM WITHOUT INTERFERENCE.

The run should either result in error messages (exit code <= 8) or should complete successfully (exit code 11). When the background job ends check the test.<name>all file carefully.

* No compiler errors or missing voter call problems (exit status > 5).

* Fatal execution time errors exit status = 7.

* Differences detected from expected output exit status =8.

II.

If the test run ends with any status but exit(11), i.e. complete success, produce an error correction request for the maintenance team by running fts_correq script. For example:

```
    fts_correq ncb2 all > correq.ncb2
```

Make sure that the number of failures you wish to analyse is set to 1. For this see fts_correq code (run fts_correq without parameters). File errdata.ncb2 will contain input data for failed cases in a form that is suitable for use by the "fts_driver.p" code in certify/. Check the content of correq.ncb2 and mail it to the maintenance team working on the <name> code (in examples: ncsuB2.i and higher versions, i.e. <name>=ncb2, or ncb3 etc).

You may also have situations where you need to send non-standard messages as part of the correction request. For example, people may send you code and reports with incorrect or inappropriate version numbers. In situations like that create and insert the message at the begining of the correq.<name> file, just after the standard initial paragraph.

III.

Now create a subdirectory that will hold the starting, and all subsequent versions for a particular program(mming team). For example:

```
    mkdir newcode/ncsuB
```

make a sub-subdirectory for the current program version:

```
    mkdir newcode/ncsuB/v2
```

and move all the files you want to keep into that sub-sub directory, e.g.

```
    mv *ncb2* newcode/ncsuB/v2
```

You may wish to use diff and compress processors to reduce stored file sizes.

Unless you are interested in doing further correlation analysis and extracting intensity functions and experimental MCF profiles (for the purpose of detecting and eliminating inter-version dependence during the acceptance testing, not assumed a standard procedure in this experiment) you may not wish to keep trace.<name>all, vect.<name>all and binrep.<name>all files. The terl.<name>all file contains a compressed overview of the executed code blocks (all begining with 0.---| have not been executed and you should find out why). You will not generate the trace, vect and terl files, nor keep binrep if you do not use the -x option. You may also wish to dispense with <name> and <name>.p files which are the executable harness+rsdimu and source harness+rsdimu respectively. We would recommend that you save at least the test.<name>all and the error.<name>all files.

Any communication (questions and answers) received prior to corrected program version are also saved into the "active" newcode sub-sub directory as "q1", "a1", "q2", "a2" etc.

IV.

Upon receiving a message with the location of the latest corrected version, and of the correction/change report(s), cd to accept/:

```
*  Create a new subsubdirectory in the appropriate program subdirectory
    e.g. ncsuB3.i location and change report have just been received

          mkdir newcode/ncsuB/v3

*  Save the location/change report message into v<number>, e.g.
   from inside the mail:

          s <#> newcode/ncsuB/v3/correction_report

          where <#> is the number of the mail message on your h-list.

*  Then (<path> points to maintenance team location the code):

     cp <path>/ncsuB3.i newcode/ncsuB/v3/ncsuB3.i

     cp <path>/ncsuB3.i ../code/ncsuB3.i

     fts_accept ncsuB3.i ncb3 all -c -x > test.ncb3all&
```

Now repeat the previous steps depending on the results of the test run, i.e. run a fts_correq if necessary, move results of the run into, for example v3 etc. Use appropriate university name and version numbers.

```
******************************************************************************
Notes:
******************************************************************************
```

It is expected that the maintenance team makes a single error correction that was requested by the correq.<name> report and sends back to you a mail message giving the location in their directories of the new and corrected code version, the new version number and one (or more if several changes had to be made to correct an error) error correction report(s). Save the received location message and the correction report into the "active" newcode sub-sub directory as "correction_report", e.g.

```
                    newcode/ncsuB/v2/correction_report
```

You procede then to pick-up the new version of the code and copy it into appropriate newcode sub-sub file, and ../code file (you may wish to use pointers/links to save space instead). Check that they send you the code and the report with an appropriate version numbers everytime.

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>                                                                        >
> Check the error report they send against the test report you have!!!   >
> If there is any indication at all that the difference may be due       >
> to an error in the "golden" code (i.e. supplied expected answers)      >
> freeze all testing and immediately inform ATS distribution site,       >
> i.e. NCSU (see fts86/ReadMe for address, phone etc.).                  >
>                                                                        >
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

It is extremely important for the success of the experiment that you keep not only the final, corrected version of each program, but that each intemediate version submitted for acceptance testing is saved and tagged with an appropriate version number and information about the changes/corrections (using the provided change form). It is expected that programmers will correct one error at a time (and should not be given requests for more than one correction at a time), so that we can keep track of the influence particular errors had on the overall system failure probability etc.

The "certify" directory contains code and files that would be sent to each maintenance/certification team. It contains a basic rsdimu driver (to avoid interface problems) and instructions on its use. It also contains a sample input and output, and an electronic error report file. Whenever a change is made in the code it is expected that the programmer will record using this report. The new version of the code and the error and change report copy are both returned to the experimenters.

It is essential that each program be given a version number and associated with it the date of it creation. Every time a program is corrected its version changes and should be recorded in the correction report, as comments in the code itself, and should reflect in the file name for the new code (as kept in the "code" directory, and in the "newcode" directory in accept/).

****

Fault-Tolerant Software Experiment

PROCEDURE FOR CERTIFICATION TESTING

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

It is assumed that the communication between the experimenters and the maintenance personnel will be via electronic mail. It is further assumed that the experimenter has full access to maintenance personnel files, but the reverse is not true. It is also assumed that the acceptance testing is performed in csh in UNIX 4.2/4.3BSD, or Ultrix 1.1/1.2 environments running on VAX hardware (else see nonVAX_host/ directory).

I.

The testing begins by placing the code which is to be tested into the code/ directory. Enter the accept/ directory and start the initial round of testing by executing the fts_certify shell script. For example:

```
fts_certify ncsuD7.i ncd7
```

or

```
fts_certify ncsuD7.i ncd7 > certify.ncd7&
```

The latter form should be used if you wish to run in the background.

NOTE: YOU CANNOT RUN TWO fts_certify JOBS FROM THE SAME DIRECTORY AT THE SAME TIME (EVEN IN BACKGROUND). THE SYSTEM WAS NOT DESIGNED FOR THAT AND YOU CAN END UP WITH A MESS. YOU CAN, HOWEVER, KEEP TWO DIRECTORIES, SAY ACCEPT1 AND ACCEPT2 AND RUN AN fts_certify FROM EACH OF THEM WITHOUT INTERFERENCE.

fts_certify calls fts_accept with all.dat testset and -c option. Output is automatically routed into file test.<name>all. This file will then contain information about the acceptance test run, and will be used as the basis for forming a correction request file correq.<name>.

The run should either result in error messages (exit code <= 8) or should complete successfully (exit code 11). When the job ends check the test.<name>all file carefully.

* No compiler errors or missing voter call problems (exit status > 5).

* Fatal execution time errors exit status = 7.

* Differences detected from expected output exit status =8.

II.

If the test run ends with any status but exit(11), i.e. complete success, an error correction request will be produced for the maintenance team. The correction request will be in correq.<name>, and the test cases that caused up to the first 20 failures will be in errdata.<name>.

Check the content of correq.<name> and mail it to the maintenance team working on the <name> code (in examples: ncsuD7.i and higher versions, i.e. <name>=ncd7, or ncd8 etc).

Mail or copy directly into the directory of the maintenace team file errdata.<name>. The format of the data in this file is suitable for direct use by their "driver" program, so that they can do their own testing.

```
*************************************************************************
*                                                                       *
*                            W A R N I N G                              *
*                                                                       *
* Your system may have a byte limit on mail messages that can be passed *
* through it (e.g. 100,000 bytes). In that case you may find that       *
* your correction report may be too large, and may become truncated     *
* by the e-mail system. It is safer to send only short messages and     *
* to transfer long files directly into certification team's directory   *
* using cp.                                                             *
*                                                                       *
* e.g. 20 failures in ncsuD7.i generate a correq.ncd7 request file of   *
*      about 4000 lines of code (about 170,000 bytes).                  *
*                                                                       *
*************************************************************************
```

You may also have situations where you need to send non-standard messages as part of the correction request. For example, people may send you code and reports with incorrect or inappropriate version numbers. In situations like that create and insert the message at the begining of the correq.<name> file, just after the standard initial paragraph.

III.

The following procedure describes program and data version management without RCS or SCCS. You should read it regardless of the management procedure you will use so that you can decide what to save/store.

Create a subdirectory that will hold the starting, and all subsequent versions for a particular program(mming team). For example:

```
    mkdir newcode/ncsuD
```

make a sub-subdirectory for the current program version:

```
    mkdir newcode/ncsuD/v7
```

and move all the files you want to keep into that sub-sub directory, e.g.

```
    mv *ncd7* newcode/ncsuD/v7
```

Unless you are interested in doing correlation analysis and extracting intensity functions and experimental MCF profiles (for the purpose of detecting and eliminating inter-version dependence during the acceptance testing, not assumed a standard procedure in this experiment) you may not wish to keep trace.<name>all, vect.<name>all and binrep.<name>all files. The terl.<name>all file contains a compressed overview of the executed code blocks (all begining with 0.---| have not been executed and you should find out why). You will not generate the trace, vect and terl files, nor keep binrep if you do not use the -x option. When using fts_certify this is default in or der to reduce program testing time and sve storage.

You may also wish to dispense with <name> and <name>.p files which are the executable harness+rsdimu and source harness+rsdimu respectively. We would recommend that you save at least the test.<name>all and the error.<name>all files.

To save space you can save only the difference in the code between the starting version and the new version (e.g. ncsuD7.i and ncsuD8.i, or ncsuD7.i and ncsuD9.i). For that purpose use the diff processor (read DIFF(1) manual).

You can further reduce storage space by "compressing" all the saved files using compress, or any other code compression processor available on your machine.

Whatever the scheme, make sure that you can rebuild the starting files and versions.

Any communication (questions and answers) received prior to corrected program version are also saved into the "active" newcode sub-sub directory as "q1", "a1", "q2", "a2" etc.

IV.

Upon receiving a message with the location of the latest corrected version, and of the correction/change report(s), cd to accept/:

* Create a new subsubdirectory in the appropriate program subdirectory e.g. ncsuD8.i location and change report have just been received

```
mkdir newcode/ncsuD/v8
```

* Save the location/change report message into v<number>, e.g. from inside the mail:

```
s <#> newcode/ncsuD/v8/correction_report
```

where <#> is the number of the mail message on your h-list.

* Then (<path> points to maintenance team location the code):

```
cp <path>/ncsuD8.i newcode/ncsuD/v8/ncsuD8.i

cp <path>/ncsuD8.i ../code/ncsuD8.i
```

alternative for latter (saves space):

```
{
    cd ../code
    ln -s ../accept/newcode/ncsuD/v8/ncsuD8.i  ncsuD8.i
}

    fts_certify ncsuD8.i ncd8 > certify.ncd8&
```

Now repeat the previous steps depending on the results of the test run.
Use appropriate university name and version numbers.

```
******************************************************************************
Notes:
******************************************************************************
```

It is expected that the maintenance team makes a error corrections for up to the first 20 reported failures that were requested by the correq.<name>. Certification (maintenance) team is supposed to mail back a message giving the location in their directories of the new and corrected code version, the new version number and one (or more if several changes had to be made) error correction report(s). You may if you wish use hardcopy correction reports, but there is a danger that the reports may evenuatly get separated from the code and corrections to which they refer. Furthermore if kept in electronic form it may be easier to analyse them.

Save the received location message and the correction report(s) into the "active" newcode sub-sub directory as "correction_report", e.g.

```
                    newcode/ncsuD/v8/correction_report
```

You procede then to pick-up the new version of the code and copy it into appropriate newcode sub-sub file, and ../code file (you may wish to use pointers/links to save space instead). Check that teams send you the code and the report with an appropriate version numbers everytime.

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>                                                                        >
> Check the error report tems send against the test report you have!!!   >
> If there is any indication at all that the difference may be due      >
> to an error in the "golden" code (i.e. supplied expected answers)      >
> freeze all testing and immediately inform ATS distribution site,       >
> i.e. NCSU (see fts87/ReadMe for address, phone etc.).                  >
>                                                                        >
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

It is extremely important for the success of the experiment that you keep not only the final, corrected version of each program, but that each intemediate version submitted for acceptance testing is saved and tagged with an appropriate version number and information about the changes/corrections (using the provided change form).

The "certify" directory contains code and files that would be sent to each maintenance/certification team. It contains a basic rsdimu driver (to avoid interface problems) and instructions on its use. It also contains a sample input and output, and an electronic error report file. Whenever a change is made in the code it is expected that the programmer will record it using this report. The new version of the code and the error and change report copy(ies) are returned to the experimenters.

It is essential that each program be given a version number and associated with it the date of its creation. Every time a program is corrected its version changes and should be recorded in the correction report, as comments in the code itself, and should reflect in the file name for the new code (as kept in the "code" directory, and in the "newcode" directory in accept/).

****

Fault-Tolerant Software Experiment

Instructions for Certification Teams

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/12-May-87

Welcome to the FTS certification project. This is a NASA sponsored experiment in Fault-Tolerant Software. Your part will be to find and correct any bugs in the software, under controlled conditions. The software means one of the twenty programs that were produced during the first phase of this experiment.

You should be in possession of the following documentation (if you are not please see your experiment supervisor).

RSDIMU specification (version 3.2/10-Feb-87)

RSDIMU specification (version 2.1/19-Sep-85)

You should also have a directory called "certify" sent to you by the FTS experimenters. In this directory you have the code for the driver, instructions for its use, and some test data. All file that are part of this testing package begin with 'fts_' and should not be modifed by you except under special circumstances, and after consultation with your experiment supervisor. You can of course copy them and then modify them at will. This is not advised.

You will be receiving messages about needed corrections via e-mail. You should read them, correct the program to the best of your abilities, and test it. The data on which the program failed will either be attached to the message or will be sent to you separately. You can then use this data to test your code.

The message with correction requests is expected to be of the form

correq.<name>

where <name> is the code for the program you are correcting (university code followed by the team code and program version number). The data for the cases you failed (suitable for use with your system are expected to in a file of the form

errdata.<name>all

Once you are satisfied that you have found the fault that is causing the error(s) you were requested to correct, you should fill out (make a copy) the error_report and send it to your supervisor's e-mail address. Your site may also require you to fill in and submit a hard copy of the report. In the same message you should also tell us what is the current version of your program and in which file one can find it (we shall need copies of your corrected programs so do not change them once you have sent a message that one is ready for pick-up).

The program which you have just finished correcting must be in a file called <unam>XX.vYY, where <unam> is the agreed upon abbreviation for the university at which the code was originally produced (e.g. ncsu, ucla, uiuc, uva), XX stands for the letter and number associated with your program code, and YY is the current version of your code (you begin by incrementing the number in XX by one). For example C6, i.e. ncsuC6.v07 means that you have updated ncsuC6 to version 7). You should also learn to update the version number in the program header in the style in which it is already there. If it is not part of the code you should add a comment header with the version number and date (e.g. ncsuC6.v07/15-Jul-86). A sample header is shown in fts_driver.p code.

If in doubt please ask about details.

Please bear in mind that the original specifications have been changed and that the latest version (the one you have) may require you not only to correct existing code, but also to add to it (for example missing calls to voter routines).

File ReadMe_data contains a description of the test cases that are being used to test your code. You should use this list in conjunction with the correq.<name> report to locate and identify errors.

Please read the documentation you have received for the experiment. Note that you should keep all communications concerning the program you have been given between yourself and the experimenter, and should communicate through electronic mail

****

e-mail address is: fts

****

Please feel free to ask e-mail questions about any part of this experiment.

Note the following rules and guidelines:

1. Do not change protections on any work-related files.

2. Communications are restricted during this experiment as follows. You may not discuss any aspects of your work in this job with other programmers. Any work-related communication between you and the Professor is to be conducted via UNIX mail. We require this so in the event of an error or ambiguity in the specifications, or some other significant event, all students may be sent a copy of the mailed question and its answer.

3. Every day you work you must log onto your UNIX account. In this way you will receive in a timely manner all mail concerning answers to questions, any updates in the specifications etc. You should also fill in a time-sheet once a day.

4. Your Professor will read the UNIX mail once early in the morning and again in the late evening (Monday through Friday). All questions should be directed to him/her.

5. It is your responsibility to read about UNIX tools you are not familiar or comfortable with.

6. Once a week, on Friday, you should submit a weekly progress report, describing the work you did during the week (number and type of errors you have corrected, any problems you have encountered running the driver harness, hardware problems, the total time you have spent working on the project during the week, whether reading or using the computer, if reading you should specify what and which part of the specs or which error prompted you to that action, etc.). Report is to be submitted via e-mail.

Good luck

Fault-Tolerant Software Experiment

Data

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

All the testcases in this release of the system comply with the specification version 3.2/10-Feb-87. A test case entry consists of an input record, and an output record. The latter contains what is believed to be the correct answer to the input record according to the 3.2 specs, and as generated/given by gold3v2.i.

Your code will be tested with a set of extremal and special value (ESV) test cases (796 of them) followed by 400 random test cases.

The ESV test cases are based on an initial random case which was then modified step by step to check a particular function and/or option. In the following overview of the ESV data only the principal features of the data sub-classes are given, along with the principal variables that were changed at any one time. Changes are given with respect to a base test case (usually #1).

1. A general random test case, DMODE=0 (RTI, foofl.dat), votecontrol=0. Checks that all the voters are off.

2-7. Test cases checking voter placement. Voters in the fts_harness are activated via VOTECONTROL one at the time. VOTECONTROL activation order 1, 3, 2, 4, 8 16.

Votecontrol and their activation results are detailed below:

1 : Sets the Linnoise values of first 4 sensors to true.
3 : Sets Linoffset value of first sensor to 0.0
2 : Sets linout value of first sensor to BADDAT;
4 : Sets SYSSTATUS to false.
8 : sets the estimate values of acceleration values to large values of 99999.0
16 : Sets garbage values in display output values.

8 General test case, (base 1) DMODE = 88.

9-19 base = case #1, systematic changes in DMODE testing for principal display modes (0,21-24,31-33,1,2,99). Check for various display modes which do valid displays, and the boundary display modes.

20-27 Check for mod 4096 (all chans), base = #1, offraw:selected values are increased with 4096 or 8192 to check if only lower order bits are being used.

28-52. base = #1, changes in DMODE and LINFAILIN, checking for different "blank" displays, specific failure display formats, and failures of one sensor (28-37), whole faces (two sensors, 38-43), and various combinations of four failing sensors (44-52), with one instance of eight failed sensors (50). The sensors are failed on input, to check for I display.

53-85. base = #1, random activation of different display modes continues (to ultimately test all values 0-99 by the end of the ESV set). Noise on calibration (OFFRAW) in steps of +/- 6, +/-12, +/- 18 and +/- 24. Case 57 test has LINSTD=8, DMODE=1 and noise on calibration channel 1 of +/- 24 (8x3=24). Also checked are the display failure formats for LINNOISE values, and the correct use of variable LINSTD, and correct computation of calibration noise levels. 6 and 24 were chosen because these were the boundary cases for noisiness for the linstd values chosen.

87-110 changes in LINSTD (9, 2,1 with +/- 24 on i/p channels) to check correct use of LINSTD variable and sensitivity of the calibration procedure.

86, 111-149. changes in RAWLIN, DMODE, LINFAILIN, various combinations of failures on input, noise and edgevector failures, base = #1. Values in RAWLIN are so changed as to reflect an assured failure in edgevector test, so that there are no ambiguities left. The values of DMODE are again chosen to test the display failure format. The failures are combined with failures on input, to see if the edgevector tests are properly employed.

150-151 Large changes in misalign [i,6] field, only the sixth axis was chosen for contamination because according to the latest specifications that is the only angle not used in the rsdimu procedure. It use significantly changes output values only if its value is much larger than normal. Changes in the values of other angles will not provided new information.

152-392. Test cases checking for the minimal sensor noise levels for failure declaration. Cases 152-365 no prior failures. Cases 366-392 prior failures on one and two faces. These test cases test the sensitivity requirements that all three edges fail the edgevector test before a failure is declared. False alarms are raised when only one or two edges fail. The normal value for the triplet threshold is 49 counts away from the correct figure for no prior failures on the rsdimu. The threshold values will change with the number and place of previous failures.

393-796 CRA proposed test cases with various combinations of sensors failed on input and up to one additional sensor failed in the edge vector test.

56 test cases with 1 sensor failed on input.
168 cases with two sensors failed on input.
120 cases with 3 sensors failed on input.
30 cases with 4 sensors failed on input.
8 cases with 7 sensors failed on input.

and the rest are other combinations.

Test cases numbers higher than 796 refer to random test cases.

Fault-Tolerant Software Experiment

DRIVER FOR THE RSDIMU CERTIFICATION

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

**** You do not need to read this if you will be using fts_compile and fts_execute macros.

fts_driver.p is a Pascal driver program to run your rsdimu procedure.You may make modifications to suit your tastes, but it is adequate in its present form. To compile it you have to include your file containing rsdimu procedure in the place provided in the source code. (It has to have a .i suffix to run successfully). Also you have to make a call to rsdimu procedure at the place designated.

The compiler command would be:

```
pc -C -s -o driver fts_driver.p
```

-s option would circumvent any problems you may encounter due to mixed letter cases and non-standard i/o handling.

The executable module is created in file "driver", which can be run as a shell command.
The driver expects the testcase input in a format as shown in the file "fts_errdata.sample". The output, after a successful run of the driver, is in fts_sample.out. Note driver is interactive.
If you wish to generate your own input data you will need to use the "No_output_data" option.

Note that there are several parameters which are special and are not part of the rsdimu variable/parameter set and are not given in the specs. These variables appear at the beginning of the fts_errdata.sample file. The rsdimu parameters begin with 15.0000 for obase. If you wish to use the fts_driver.p on its own and without golden data then you need to retain only the line before 15.0000 (votecontrol, case number). Votecontrol serves to control special voter routine actions (whether a particular voter changes the values of it parameters or not). It is used solely for testing placement and use of the voter routines. You need to leave it as is for regression testing of your code after correction. You may experiment with it if you wish to build you own test sets. You do not have to worry about it in the rsdimu code, the variable is taken care of in the driver code.

The other parameters control the comparisons with golden answer and you do not need to use them, unless you provide full format of the file (with dummy golden answers for example).

Fault-Tolerant Software Experiment

Testing RSDIMU code

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

Using the fts_compile and fts_execute macros is simple. Run them without any parameters to obtain the description of the paramters you need. The following sample should help.

It is assumed that you have received correq.ncd6 and errdata.ncd6all from your experiment supervisor.

To compile program ncsuD7 which is (let's assume so) in your certify file, and is the program you have just corrected run

```
fts_compile ncsuD7 7 > c.7&
```

When the run finishes check c.7 for compilation errors etc. If ok proceed (rsdimu.7 will contain driver+ncsuD7 executable code).

```
fts_execute 7 errdata.ncd6all ncsuD7 > x.7&
```

When job finishes check x.7. If there still are differences from the expected outputs go back and correct your code once more, otherwise submit error_reports and the new code to your supervisor.

Make use of the correq.ncd6 and ReadMe_data.

Fault-Tolerant Software Experiment

DATA FOR THE ACCEPTANCE TESTING

RSDIMU ACCEPTANCE TESTING SYSTEM (RSDIMU-ATS)

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

All the testcases in this release of the system comply with the specification version 3.2/10-Feb-87. A test case entry consists of an input record, and an output record. The latter contains what is believed to be the correct answer to the input record according to the 3.2 specs, and as generated/given by gold3v2.i.

The test cases are supplied in Pascal readable files.
The format can be found in the fts_prnt.p source code in accept/.
The format in which the test cases are given is suitable for use with the accept/fts_accept. Use of the system in non-Vax and UNIX-like environments is described in nonVAX_host directory, data may be re-generated using code suplied in the "generators" directory.
The latter action is should not be undertaken without consultation with the ATS distribution site (NCSU).

If you wish to print out all, or some, of the test cases use accept/fts_listdata.

If you wish to compare (difference) test cases use fts_diff.

This set of test cases was designed and generated for acceptance testing of the rsdimu code. It consists of a group of 796 extremal/special value (ESV) test cases and a group of 400 random test cases. There are four files of data:

all.dat - ESV test cases, followed by random test cases (randomNCSU.dat, then randomCRA.dat).

esv.dat - ESV test cases, only (796).

randNCSU.dat - random test cases, only (uniform sampling, 200).

randCRA.dat - random test cases, only (shaped sampling, 200).

A successful pass through all the test cases gives an estimated lower limit on the reliability of the rsdimu code of about 0.992 (valid for the employed sampling profile).

The all.dat set should provide 100% block coverage of the rsdimu code. If this is not the case (running fts_accept with -x option will give the coverage info), one should very carefully examine the tested code in places where coverage was not provided. The nature of the rsdimu problem, and the specifications, is such that a thorough programmer can provide for situations and functions which are not explicitly handled in the specifications (e.g. singular matrices, large changes in the slope constants leading to large raw acceleration values). Redundant code of the type that cannot be excited according to the current specifications, but could possibly be needed under exceptional circumstances, should be tested by the programmers providing it. They should also provide test cases for these situations (if possible). Alternatively they should provide a written explanation of the cirumstances and reasons for including that particular code. The golden program gold3v1.i, for example has 5 blocks handling display of extremal input acceleration values (>10g) which are not tested by the current acceptance data set since such large input values are outside the conversion range of the provided equations.

The coverage figures should be considered only in the last stage of the acceptance testing, i.e. when all.dat cases have been passed without a failure, and all the corrections requests have been implemented (e.g. after the final regression pass through all.dat).

The ESV data set is further described in the ReadMe_esv file, and the random data sets are described in the ReadMe_random file.

Fault-Tolerant Software Experiment

THE ESV DATA FOR THE ACCEPTANCE TESTING

RSDIMU ACCEPTANCE TESTING SYSTEM (RSDIMU-ATS)

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

The data file esv.dat contains 796 extremal and special (ESV) test cases. The test cases were designed to provide full functional coverage of the RSDIMU specifications v3.2/10-Feb-87.

The test cases are based on an initial random case which was then modified step by step to check a particular function and/or option. In the following overview of the ESV data only the principal features of the data sub-classes are given, along with the principal variables that were changed at any one time. Changes are given with respect to a base test case (usually #1).
Listing of all or some of the test cases can be obtained by running accept/fts_listdata. Difference between test cases may be examined using fts_diff.

1. A general random test case, DMODE=0 (RTI, foofl.dat), votecontrol=0. Checks that all the voters are off.

2-7. Test cases checking voter placement. Voters in the fts_harness are activated via VOTECONTROL one at the time. VOTECONTROL activation order 1, 3, 2, 4, 8 16.

Votecontrol and their activation results are detailed below:

1 : Sets the Linnoise values of first 4 sensors to true.
3 : Sets Linoffset value of first sensor to 0.0
2 : Sets linout value of first sensor to BADDAT;
4 : Sets SYSSTATUS to false.
8 : sets the estimate values of acceleration values to large values of 99999.0
16 : Sets garbage values in display output values.

8 General test case, (base 1) DMODE = 88.

9-19 base = case #1, systematic changes in DMODE testing for principal display modes (0,21-24,31-33,1,2,99).
Check for various display modes which do valid displays, and the boundary display modes.

20-27 Check for mod 4096 (all chans), base = #1, offraw:selected values are increased with 4096 or 8192 to check if only lower order bits are being used.

28-52. base = #1, changes in DMODE and LINFAILIN, checking for different "blank" displays, specific failure display formats, and failures of one sensor (28-37), whole faces (two sensors, 38-43), and various combinations of four failing sensors (44-52), with one instance of eight failed sensors (50). The sensors are failed on input, to check for I display.

53-85. base = #1, random activation of different display modes continues (to ultimately test all values 0-99 by the end of the ESV set). Noise on calibration (OFFRAW) in steps of +/- 6, +/-12, +/- 18 and +/- 24. Case 57 test has LINSTD=8, DMODE=1 and noise on calibration channel 1 of +/- 24 (8x3=24). Also checked are the display failure formats for LINNOISE values, and the correct use of variable LINSTD, and correct computation of calibration noise levels. 6 and 24 were chosen because these were the boundary cases for noisiness for the linstd values chosen.

87-110 changes in LINSTD (9, 2,1 with +/- 24 on i/p channels) to check correct use of LINSTD variable and sensitivity of the calibration procedure.

86, 111-149. changes in RAWLIN, DMODE, LINFAILIN, various combinations of failures on input, noise and edgevector failures, base = #1. Values in RAWLIN are so changed as to reflect an assured failure in edgevector test, so that there are no ambiguities left. The values of DMODE are again chosen to test the display failure format. The failures are combined with failures on input, to see if the edgevector tests are properly employed.

150-151 Large changes in misalign [i,6] field, only the sixth axis was chosen for contamination because according to the latest specifications that is the only angle not used in the rsdimu procedure. It use significantly changes output values only if its value is much larger than normal. Changes in the values of other angles will not provied new information.

152-392. Test cases checking for the minimal sensor noise levels for failure declaration. Cases 152-365 no prior failures. Cases 366-392 prior failures on one and two faces. These test cases test the sensitivity requirements that all three edges fail the edgevector testi before a failure is declared. False alarms are raised when only one or two edges fail. The normal value for the triplet threshold is 49 counts away from the correct figure for no prior failures on the rsdimu. The threshold values will change with the number and place of previous failures.

393-796 CRA proposed test cases with various combinations of sensors failed on input and up to one additional sensor failed in the edge vector test.

```
          56 test cases with 1 sensor failed on input.
         168 cases with two sensors failed on input.
         120 cases with 3 sensors failed on input.
          30 cases with 4 sensors failed on input.
           8 cases with 7 sensors failed on input.
```

and the rest are other combinations.

More detailed information about the ESV test cases can be obtained by displaying the differences between a chosen base case (#1 usually) and a series of other test cases. Utility shell script fts_diff, based on the UNIX diff processor, is provided for this purpose. By executing

```
               fts_diff esv.dat esvdiff 115 123 1
```

you can obtain, for example, in file esvdiff differences in the input values of cases 115 to 123 with respect to test case #1 of the data file esv.dat.

The CRA document regarding choice of random and ESV test cases was provided as a separate item (not in electronic form) with release 2.0 of RSDIMU-ATS.

Fault-Tolerant Software Experiment

THE RANDOM DATA FOR THE ACCEPTANCE TESTING

RSDIMU ACCEPTANCE TESTING SYSTEM (RSDIMU-ATS)

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

This set of 400 random test cases for rsdimu code is provided primarily for the purpose of estimating the lower limit on the reliability of the tested code, and as a check on the completeness of the ESV test cases. The test cases are completely independent, and no attempt was made to mimic a flight trajectory and the associated time correlation among the input variable values. Therefore any cumulative effects linked to time correlation or auto-correlation remain untested, here and in the extremal and special value (esv.dat) set.

The random data are provide in two sub-sets: randomNCSU.dat and randomCRA.dat.

****

randomNCSU.dat

Within the employed sampling domain the distribution of the generated input values is essentially flat. It contains 200 test cases.

In all cases the random data were generated using the random number generator provided with the Pascal comiler (pc, UNIX/Ultrix). Details of the mapping from the random numbers into the actual input variables are given below. More details will be supplied on request. The part of the code used to generate random input values is also enclosed (as fts_NCSUzzrand.i), and it derives in part from the RTI random test harness (September 85).

The input variables randomly sampled, or computed on the basis of randomly sampled values are:

offraw, linfailin, rawlin, misalign, normface, temp, phiv, thetav, psiv, phi, thetai, psii, dmode, linnoise, linfailout, scale0,1,2, obase, linstd and nsigt.

A more thorugh understanding of the random generation process and of the resulting input profiles can be gained by studying the fts_NCSUzzrand.i code.

The random set, randomNCSU.dat, consists of two hundred random test cases stratified into two sub-sets. The first one hundred test cases have the noise on sensors (rawlin) boosted by 200 counts everytime linfailout for a sensor is true. Thus the sensor noise level is guaranteed to exceed the sensitivity threshold of about 50 counts and the sensor should be recognized as failed. The second one hundred test cases, on the other hand, have the noise added as a uniform distribution between 1 and maxnoise-1 counts, and at half the uniform frequency for 0 and maxnoise, the latter value having been read in by the driver program. In this particular case maxnoise was 110, therefore the added noise was symmetrically centered around the threshold value of 55 counts.

It is important to note that random test cases are intended to run after all ESV test cases have been successfully negotiated. There are special situations and combinations of variable values that are covered in ESV test cases and not covered by the sampling domain used to generate present random test cases. Our experience with the random testing of rsdimu code is that the sensitivity of the random test cases to errors is very low. Unless very detailed partitioning is employed (better to use ESV cases in that case) detection capabilities of the random test cases to distinct errors saturate extremely quickly. After 2 to 10 random test cases the same errors are usually detected over and over again (if not removed). Once past 100 random test cases detection of new, different, errors becomes an almost negligible event, unless the random sampling profile is changed and tuned to the character of the already detected faults, or partitions not previously covered are sampled.

For all practical purposes the two sets of 100 test cases, are a single random set of 200 test cases, which if executed successfully, provides us with a lower limit for the rsdimu reliability (at the 95% confidence level) of about 0.985.

initial random seed for 1st 100 cases is: 777

initial random seed for 2nd 100 cases is: 1234567890

****

randomCRA.dat

The second sub-set, randCRA.dat, was generated on the basis of the CRA document TM8602/26-Aug-86.

> CRA random test cases are generated with the specifications provided in the CRA documents (especially for PHIV, PSIV and THETAV, NSIGT = 2..7 etc). The calibration noise is normally distributed, and the number of noisy sensors during calibration is exponentially distributed with a parameter of 0.18. The edge vector test can fail one additional sensor, with random noise of upto 200 counts. No sensors fail on input. Generation details can be found in fts_CRAzzrand.i

initial random seed is: 987654321

****

For all practical purposes the two sets of 200 test cases, are a single random set of 400 test cases, which if executed successfully, provides us with a lower limit for the rsdimu reliability (at the 95% confidence level) of about 0.992.

During the generation of the random test cases care is taken to examine the obtained data and to eliminate cases where more than one sensor fails in flight.

Fault-Tolerant Software Experiment

THE GOLDEN DISPLAY AND RSDIMU CODES

RSDIMU-ATS 3.0/PR/UNIX/FTS.NASA-LaRC.Va/NCSU.CSC/05-Mar-87

There are 2 files in this directory viz. display.i, and gold3v1.i The sources correspond to specification version v3.2/10-Feb-87,

display.i contains the display module extracted from the gold program. It has been extensively tested as a stand alone module, and gives results in accordance with the specifications v3.2. It does not need any special declarations in the main program except for those which are in the RSDIMU procedure assumed to be globally available (only the type declarations.) To use this procedure just use standard #include compiler option, and the calling format

```
   display (DISMODE, DISUPPER, DISLOWER);
```

Use of the golden rsdimu follows the same rules as use of any other rsdimu code, and is fully explained in the specs (see also certify/driver.p).