The Alog Toolbox Command Line Utilities

June 2018

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139
`project-pavlab/apps/app_alog_cmdline`

1 Overview

The Alog Toolbox, in addition to the `alogview` GUI based utility, also contains a set of command line post-mission analysis utility applications:

- `alogclip`
- `aloggrep`
- `alogrm`
- `alogscan`
- `aloghelm`
- `alogiter`
- `alogsplit`
- `alogpare`
- `alogcd`
- `alogcat`
- `alogavg`
- `alogmhash`
- `alognpos`

Each application manipulates or renderings `.alog` files generated by the `pLogger` application. Four

of the applications, alogclip, aloggrep, alogpare, and alogrm are command-line tools for filtering a given .alog file to a reduced size. Reduction of a log file size may facilitate the time to load a file in a post-processing application, may facilitate its transmission over slow transmission links when analyzing data between remote users, or may simply ease in the storing and back-up procedures. The alogscan tool provides statistics on a given .alog file that may indicate how to best reduce file size by eliminating variable entries not used in post-processing. It also generates other information that may be handy in debugging a mission. The alogsplit tool will split a single alog file into a folder containing a dedicated alog file for each logged variable. This operation is also done automatically upon the launch of appalogview when launched on an alog file for the first time. The alogview tool is a GUI-based tool that accepts one or more .alog files and renders a vehicle positions over time on an operation area, provides time-correlated plots of any logged numerical MOOS variables, and renders helm autonomy mode data with plots of generated objective functions.

2 An Example .alog File

The .alog file used in the examples below was generated from the Alpha example mission. This file, alpha.alog, is found in the missions distributed with the MOOS-IvP tree. The alpha.alog file was created by simply running the mission as described, and can be found in:

```
moos-ivp/trunk/ivp/missions/alpha/alpha.alog
```

3 The alogscan Tool

The alogscan tool is a command-line application for providing statistics relating to a given .alog file. It reports, for each unique MOOS variable in the log file, (a) the number of lines in which the variable appears, i.e., the number of times the variable was posted by a MOOS application, (b) the total number of characters comprising the variable value for all entries of a variable, (c) the timestamp of the first recorded posting of the variable, (d) the timestamp of the last recorded posting of the variable, (e) the list of MOOS applications the posted the variable.

3.1 Command Line Usage for the alogscan Tool

The alogscan tool is run from the command line with a given .alog file and a number of options. The usage options are listed when the tool is launched with the -h switch:

```
$ alogscan --help or -h
```

```
 1  Usage:
 2    alogscan file.alog [OPTIONS]
 3
 4  Synopsis:
 5    Generate a report on the contents of a given
 6    MOOS .alog file.
 7
 8  Options:
 9  --sort=type   Sort by one of SIX criteria:
10                start: sort by first post of a var
11                stop:  sort by last post of a var
12     (Default)  vars:  sort by variable name
13                proc:  sort by process/source name
14                chars: sort by total chars for a var
15                lines: sort by total lines for a var
16
17  --appstat     Output application statistics
18  -r,--reverse  Reverse the sorting output
19  -n,--nocolors Turn off process/source color coding
20  -h,--help     Displays this help message
21  -v,--version  Displays the current release version
22  --rate_only   Only report the data rate
23  --noaux       Ignore auxilliary source info
24
25  See also: aloggrp, alogrm, alogclip, alogview
```

The order of the arguments passed to `alogscan` do not matter. The lines of output are sorted by grouping variables posted by the same MOOS process or source. The sorting criteria can instead be done by alphabetical order on the variable name (`--sort=vars`), the total characters in the file due to a variable (`--sort=chars`), the total lines in the file due to a variable (`--sort=lines`), the time of the first posting of the variable (`--sort=start`), or the time of the last posting of the variable (`--sort=stop`). The order of the output may be reversed (`-r, --reverse`). By default, the entries are color-coded by the variable source, using the few available terminal colors (there are not many). When unique colors are exhausted, the color reverts back to the default terminal color in effect at the time.

3.2 Example Output from the alogscan Tool

The output shown below was generated from the `alpha.alog` file generated by the Alpha example mission.

```
$ alogscan file.alog
```

```
Variable Name           Lines   Chars   Start    Stop  Sources
-------------           -----   -----  ------  ------  -------
DB_CLIENTS                282   22252   -0.38  566.42  MOOSDB_alpha
DB_TIME                   556    7132    1.21  566.18  MOOSDB_alpha
DB_UPTIME                 556    7173    1.21  566.18  MOOSDB_alpha
USIMMARINE_STATUS         276   92705    0.39  565.82  uSimMarine
NAV_DEPTH                6011    6011    1.43  566.38  uSimMarine
NAV_HEADING              6011   75312    1.43  566.38  uSimMarine
NAV_LAT                  6011   74799    1.43  566.38  uSimMarine
NAV_LONG                 6011   80377    1.43  566.38  uSimMarine
NAV_SPEED                6011    8352    1.43  566.38  uSimMarine
NAV_STATE                6011   18033    1.43  566.38  uSimMarine
NAV_X                    6011   72244    1.43  566.38  uSimMarine
NAV_Y                    6011   77568    1.43  566.38  uSimMarine
NAV_YAW                  6011   80273    1.43  566.38  uSimMarine
BHV_IPF                  2009  564165   46.26  542.85  pHelmIvP
CREATE_CPU               2108    2348   46.26  566.33  pHelmIvP
CYCLE_INDEX                 5       5   44.98  543.09  pHelmIvP
DEPLOY                      3      14    3.84  543.09  pHelmIvP,pMarineViewer
DESIRED_HEADING          2017    5445    3.85  543.09  pHelmIvP
DESIRED_SPEED            2017    2017    3.85  543.09  pHelmIvP
HELM_IPF_COUNT           2108    2108   46.26  566.32  pHelmIvP
HSLINE                      1       3    3.84    3.84  pHelmIvP
IVPHELM_DOMAIN              1      29    3.84    3.84  pHelmIvP
IVPHELM_ENGAGED           462    3342    3.85  566.32  pHelmIvP
IVPHELM_MODESET             1       0    3.84    3.84  pHelmIvP
IVPHELM_POSTINGS         2014  236320   46.26  543.33  pHelmIvP
IVPHELM_STATEVARS           1      20   44.98   44.98  pHelmIvP
IVPHELM_SUMMARY          2113  612685   44.98  566.33  pHelmIvP
LOOP_CPU                 2108    2348   46.26  566.33  pHelmIvP
PC_hsline                   1       9   44.98   44.98  pHelmIvP
PC_waypt_return             3      14   44.98  543.33  pHelmIvP
PC_waypt_survey             3      14   44.98  543.33  pHelmIvP
PHELMIVP_STATUS           255  198957    3.85  565.12  pHelmIvP
PLOGGER_CMD                 1      17    3.84    3.84  pHelmIvP
PWT_BHV_HSLINE              1       1   44.98   44.98  pHelmIvP
PWT_BHV_WAYPT_RETURN        3       5   44.98  543.09  pHelmIvP
PWT_BHV_WAYPT_SURVEY        2       4   44.98  462.90  pHelmIvP
RETURN                      4      19    3.84  543.09  pHelmIvP,pMarineViewer
STATE_BHV_HSLINE            1       1   44.98   44.98  pHelmIvP
STATE_BHV_WAYPT_RETURN      4       4   44.98  543.33  pHelmIvP
STATE_BHV_WAYPT_SURVEY      3       3   44.98  463.15  pHelmIvP
SURVEY_INDEX               10      10   44.98  429.70  pHelmIvP
SURVEY_STATUS            1116   77929   45.97  462.90  pHelmIvP
VIEW_POINT               4034  101662   44.98  543.33  pHelmIvP
VIEW_SEGLIST                4     273   44.98  543.33  pHelmIvP
WPT_INDEX                   1       1  463.15  463.15  pHelmIvP
WPT_STAT                  223   15626  463.15  543.09  pHelmIvP
LOGGER_DIRECTORY           56    1792    1.07  559.19  pLogger
PLOGGER_STATUS            263  331114    1.07  566.40  pLogger
DESIRED_RUDDER          10185  150449   -9.28  545.18  pMarinePID
DESIRED_THRUST          10637   20774   -9.28  566.52  pMarinePID
MOOS_DEBUG                  5      39   -9.31  545.23  pMarinePID,pHelmIvP
PMARINEPID_STATUS         279   81990    0.95  566.28  pMarinePID
HELM_MAP_CLEAR              1       1   -1.56   -1.56  pMarineViewer
MOOS_MANUAL_OVERIDE         1       5   44.65   44.65  pMarineViewer
PMARINEVIEWER_STATUS      270   95560   -0.95  564.89  pMarineViewer
NODE_REPORT_LOCAL        1159  207535    1.15  565.91  pNodeReporter
PNODEREPORTER_STATUS      233   50534   -0.37  563.93  pNodeReporter
-----------------------------------------------------------------
Total variables: 57
Start/Stop Time: -9.31 / 566.52
```

When the -appstat command line option is included, a second report is generated, after the above report, that provides statistics keyed by application, rather than by variable. For each application that has posted a variable recorded in the given .alog file, the number of lines and characters are recorded, as well as the percentage of total lines and characters. An example of this report:

MOOS Application	Total Lines	Total Chars	Lines/Total	Chars/Total
MOOSDB_alpha	1394	36557	1.37	1.08
uSimMarine	54375	585674	53.57	17.29
pHelmIvP	22642	1825437	22.31	53.89
pLogger	319	332906	0.31	9.83
pMarinePID	21106	253252	20.80	7.48
pMarineViewer	279	95599	0.27	2.82
pNodeReporter	1392	258069	1.37	7.62

Further Tips

- If a small number of variables are responsible for a relatively large portion of the file size, and are expendable in terms of how data is being analyzed, the variables may be removed to ease the handling, transmission, or storage of the data. To remove variables from existing files, the alogrm tool described in Section 6 may be used. To remove the variable from future files, the pLogger configuration may be edited by either removing the variable from the list of variables explicitly requested for logging, or if WildCardLogging is used, mask out the variable with the WildCardOmitPattern parameter setting. See the pLogger documentation.
- The output of alogscan can be further distilled using common tools such as grep. For example, if one only wants a report on variables published by the pHelmIvP application, one could type:

```
$ alogscan alpha.alog | grep pHelmIvP
```

4 The alogclip Tool

The alogclip tool will prune a given .alog file based on a given beginning and end timestamp. This is particularly useful when a log file contains a sizeable stretch of data logged after mission completion, such as data being recorded while the vehicle is being recovered or sitting idle topside after recovery.

4.1 Command Line Usage for the alogclip Tool

The alogclip tool is run from the command line with a given .alog file, a start time, end time, and the name of a new .alog file. By default, if the named output file exists, the user will be prompted before overwriting it. The user prompt can be bypassed with the -f,--force option. The usage options are listed when the tool is launched with the -h switch:

```
$ alogclip --help or -h
```

```
Usage:
  alogclip in.alog mintime maxtime [out.alog] [OPTIONS]

Synopsis:
  Create a new MOOS .alog file from a given .alog file
  by removing entries outside a given time window.

Standard Arguments:
  in.alog  - The input logfile.
  mintime  - Log entries with timestamps below mintime
             will be excluded from the output file.
  maxtime  - Log entries with timestamps above mintime
             will be excluded from the output file.
  out.alog - The newly generated output logfile. If no
             file provided, output goes to stdout.

Options:
  -h,--help     Display this usage/help message.
  -v,--version  Display version information.
  -f,--force    Overwrite an existing output file
  -q,--quiet    Verbose report suppressed at conclusion.

Further Notes:
  (1) The order of arguments may vary. The first alog
      file is treated as the input file, and the first
      numerical value is treated as the mintime.
  (2) Two numerical values, in order, must be given.
  (3) See also: alogscan, alogrm, aloggrep, alogview
```

4.2 Example Output from the alogclip Tool

The output shown below was generated from the `alpha.alog` file generated by the Alpha example mission.

```
$ alogclip alpha.alog new.alog 50 350
```

```
Processing input file alpha.alog...

Total lines clipped:      44,988  (44.32 pct)
  Front lines clipped:     5,474
  Back  lines clipped:    39,514
Total chars clipped:   4,200,260  (43.09 pct)
  Front chars clipped:   432,409
  Back  chars clipped: 3,767,851
```

5 The aloggrep Tool

The aloggrep tool has two primary purposes. The first is to prune a given `.alog` file into a smaller, yet syntactically valid `.alog` file. It this mode it can also be used for enforcing strict ordering of entries, and removing duplicate entries, since the pLogger app does not guarantee perfect ordering or files without the occasional duplicate entry. The second primary purpose is to extract data for

plotting in a tool like matlab or similar, or extracting a single value that may be ingested in a shell script. These use cases are described below.

A concise form of this documentation is available with `aloggrep --help`, and the web version of this content can be quickly opened with `aloggrep --web`.

5.1 Using aloggrep to Produce a Reduced and/or Ordered Output

The aloggrep tool will prune a given `.alog` file by retaining lines of the original file that contain log entries for a user-specified list of MOOS variables or MOOS processes (sources). As the name implies it is motivated by the Unix `grep` command, but `grep` will return a matched line *regardless* of where the pattern appears in the line. Since MOOS variables also often appear in the string content of other MOOS variables, `grep` often returns much more than one is looking for. The aloggrep tool will only pattern-match on the second column of data (the MOOS variable name), or the third column of data (the MOOS source), of any given entry in a given `.alog` file.

For example, try running the alpha mission and let the vehicle traverse the waypoints for a little while. Quit the mission and find the `.alog` file.

```
$ cd alpha_4_1_2022_____18_33_46/
$ ls
alpha._bhv                          alpha_4_1_2022_____18_33_46.blog
alpha_4_1_2022_____18_33_46._moos   alpha_4_1_2022_____18_33_46.xlog
alpha_4_1_2022_____18_33_46.alog    alpha_4_1_2022_____18_33_46.ylog
```

Then we can use aloggrep to see what is happening with the DEPLOY and WFLAG variable. The former is used for starting the mission, while the latter is incremented each time the vehicle hits a waypoint:

```
$ aloggrep DEPLOY WFLAGalpha_4_1_2022_____18_33_46.alog
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:       ./alpha_4_1_2022_____18_33_46/alpha_4_1_2022_____18_33_46.alog
%% FILE OPENED ON  Wed Dec 31 19:00:00 1969
%% LOGSTART           41033480654.29403
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26.28008        DEPLOY               pHelmIvP:HELM_VAR_INIT false
26.28008        DEPLOY               pHelmIvP:HELM_VAR_INIT false
26.28008        DEPLOY               pHelmIvP:HELM_VAR_INIT false
26.28008        DEPLOY               pHelmIvP:HELM_VAR_INIT false
26.28008        DEPLOY               pHelmIvP:HELM_VAR_INIT false
26.28008        DEPLOY               pHelmIvP:HELM_VAR_INIT false
74.65635        DEPLOY               pMarineViewer   true
74.65635        DEPLOY               pMarineViewer   true
74.65635        DEPLOY               pMarineViewer   true
85.23170        WFLAG                pHelmIvP:38:waypt_survey waypoints=1
116.06017       WFLAG                pHelmIvP:148:waypt_survey waypoints=2
74.65635        DEPLOY               pMarineViewer   true
116.06017       WFLAG                pHelmIvP:148:waypt_survey waypoints=2
140.11575       WFLAG                pHelmIvP:234:waypt_survey waypoints=3
158.09377       WFLAG                pHelmIvP:298:waypt_survey waypoints=4
176.15134       WFLAG                pHelmIvP:362:waypt_survey waypoints=5
74.65635        DEPLOY               pMarineViewer   true
176.15134       WFLAG                pHelmIvP:362:waypt_survey waypoints=5
199.22812       WFLAG                pHelmIvP:444:waypt_survey waypoints=6
230.22727       WFLAG                pHelmIvP:554:waypt_survey waypoints=7
254.00302       WFLAG                pHelmIvP:640:waypt_survey waypoints=8
  Total lines retained: 21 (0.05%)
  Total lines excluded: 44238 (99.95%)
  Total chars retained: 1435 (0.04%)
  Total chars excluded: 3873898 (99.96%)
    Variables retained: (2) DEPLOY, WFLAG
```

The last five lines provide a summary contained retained and excluded. All lines above that are essentially a valid `.alog` file. Note however that there are some duplicate lines, and a couple lines out of order. To order the output and remove duplicates, try running with the `--sort,-s` and `--duplicates,-d` options, or simply the `-sd` option to do both:

```
$ aloggrep DEPLOY WFLAG alpha_4_1_2022_____18_33_46.alog -sd
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:       ./alpha_4_1_2022_____18_33_46/alpha_4_1_2022_____18_33_46.alog
%% FILE OPENED ON  Wed Dec 31 19:00:00 1969
%% LOGSTART           41033480654.29403
26.28008        DEPLOY               pHelmIvP:HELM_VAR_INIT false
74.65635        DEPLOY               pMarineViewer   true
85.23170        WFLAG                pHelmIvP:38:waypt_survey waypoints=1
116.06017       WFLAG                pHelmIvP:148:waypt_survey waypoints=2
140.11575       WFLAG                pHelmIvP:234:waypt_survey waypoints=3
158.09377       WFLAG                pHelmIvP:298:waypt_survey waypoints=4
176.15134       WFLAG                pHelmIvP:362:waypt_survey waypoints=5
199.22812       WFLAG                pHelmIvP:444:waypt_survey waypoints=6
230.22727       WFLAG                pHelmIvP:554:waypt_survey waypoints=7
254.00302       WFLAG                pHelmIvP:640:waypt_survey waypoints=8
  Total re-sorts: 2
  Total lines retained: 10 (0.02%)
  Total lines excluded: 44238 (99.98%)
  Total chars retained: 723 (0.02%)
  Total chars excluded: 3873898 (99.98%)
    Variables retained: (2) DEPLOY, WFLAG
```

Notice the duplicates are removed, and the entries are ordered. Note also that, with alog tools, the order of the arguments on the command line do not matter. In the above example, the report section on the bottom now also indicates how many lines needed to be re-sorted, in this case 2.

5.1.1 Creating a New ALog File

Notice that the output above is not quite a syntactically valid `.alog` file since the report lines at the end are not log lines. So if output is simply re-directed to a file, the following would produce an `.alog` file with essentially garbage at the end:

```
$ aloggrep DEPLOY WFLAG alpha_4_1_2022_____18_33_46.alog -sd > newfile.alog
```

To create a new logfile either suppress the report lines at the end with:

```
$ aloggrep DEPLOY WFLAG oldfile.alog -sd --no_report > newfile.alog
```

Or simply provide a second (new) file name and the new file will not include the report lines:

```
$ aloggrep DEPLOY WFLAG alpha_4_1_2022_____18_33_46.alog -sd  newfile.alog
  Total re-sorts: 2
  Total lines retained: 10 (0.02%)
  Total lines excluded: 44238 (99.98%)
  Total chars retained: 723 (0.02%)
  Total chars excluded: 3873898 (99.98%)
    Variables retained: (2) DEPLOY, WFLAG
```

Note the report lines still go to the terminal, but the log files are written to the new `.alog` file.

5.1.2 Filtering based on the Data Source (Publishing App)

Rather than naming variables to keep, `aloggrep` can be provided with the name of one or more apps, and all entries published by these apps will be retained.

```
$ aloggrep pHelmIvP alpha_4_1_2022_____18_33_46.alog
```

5.1.3 Wildcard Matching

Limited simple wildcard matching is supported:

```
$ aloggrep NAV_*  file.alog
```

The above will grab all variables beginning with `NAV_`. Placing the asterisk in any other position will have no effect, e.g., `*_SPEED` will not grap `NAV_SPEED`.

5.2 Using aloggrep to Extract Plottable Data

An additional use case for `aloggrep` involves getting data for plotting. Using the same example as above, suppose we want to plot the value of `WPT_ODO` versus time. This variable represents vehicle odometry data, published by the waypoint behavior, and reset to zero on each waypoint. Using the above arguments we get:

```
$ aloggrep alpha_4_1_2022_____18_33_46.alog WPT_ODO
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% LOG FILE:       ./alpha_4_1_2022_____18_33_46/alpha_4_1_2022_____18_33_46.alog
%% FILE OPENED ON  Wed Dec 31 19:00:00 1969
%% LOGSTART           41033480654.29403
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
74.83427        WPT_ODO        pHelmIvP:1:waypt_survey   0.00000
75.90277        WPT_ODO        pHelmIvP:5:waypt_survey   0.34869
76.20800        WPT_ODO        pHelmIvP:6:waypt_survey   0.57995
76.49590        WPT_ODO        pHelmIvP:7:waypt_survey   0.88998
76.77897        WPT_ODO        pHelmIvP:8:waypt_survey   1.42170
77.08220        WPT_ODO        pHelmIvP:9:waypt_survey   2.79976
77.39445        WPT_ODO        pHelmIvP:10:waypt_survey  3.53386
77.67854        WPT_ODO        pHelmIvP:11:waypt_survey  4.38950
77.95012        WPT_ODO        pHelmIvP:12:waypt_survey  5.40420

...

  Total lines retained: 571 (1.29%)
  Total lines excluded: 43688 (98.71%)
  Total chars retained: 43305 (1.12%)
  Total chars excluded: 3832028 (98.88%)
    Variables retained: (1) WPT_ODO
```

The above output does grab the relevant lines, but we'd like to drop the header lines and report at

the end. Using the `--format=time:val` of `--tv` option, the results are stripped down to data lines only and just the time and value columns are retained:

```
$ aloggrep alpha_4_1_2022_____18_33_46.alog WPT_ODO --format=time:val
74.83427 0.00000
75.90277 0.34869
76.20800 0.57995
76.49590 0.88998
76.77897 1.42170
77.08220 2.79976
77.39445 3.53386
77.67854 4.38950
77.95012 5.40420
...
```

5.2.1 Extracting Plottable Data from a Complex Posting

Not all posted data is in simple numerical format as in case above with the variable `WPT_ODO`. Suppose for example, instead of `WPT_ODO`, our objective is to plot the value of the waypoint index versus time. This value, in the Alpha mission, is published as part of a string:

```
$ aloggrep alpha_4_1_2022_____18_33_46.alog WFLAG --format=time:val
85.23170 waypoints=1
116.06017 waypoints=2
140.11575 waypoints=3
158.09377 waypoints=4
176.15134 waypoints=5
199.22812 waypoints=6
230.22727 waypoints=7
254.00302 waypoints=8
```

In this case, we would like to isolate the numerical value from the string. Using the `--subpat=PATTERN` option we can isolate numerical values from variable postings made in the common format of:

```
field1=value1, field2=value2, ..., fieldN=valueN
```

```
$ aloggrep alpha_4_1_2022_____18_33_46.alog WFLAG --format=time:val --subpat=waypoints
85.23170 1
116.06017 2
140.11575 3
158.09377 4
176.15134 5
199.22812 6
230.22727 7
254.00302 8
```

Note in the above example, the column separator is single white space character (ASCII 32). If a colon separator is desired, use the `--cso` flag. If a comma separator is desired, use the `--csc` flag.

5.3 Extracting a First or Final Posting

In some cases, aloggrep may be used as a tool within a script to determine mission outcome. If the mission outcome is represented in a single MOOS variable, e.g., MISSION_RESULT or MISSION_SCORE, then the goal is to isolate just this value. Using the --first or --final option, we can reduce the output to just one line. Using the example above:

```
$ aloggrep alpha_4_1_2022_____18_33_46.alog WFLAG --format=time:val --final
254.00302 8
```

To isolate just the value column, use the --format=val, or --v option:

```
$ aloggrep alpha_4_1_2022_____18_33_46.alog WFLAG --v --final
8
```

The style of output can then be used within a shell script to take action based on the result, stored in a shell variable:

```
$ FOO=(‘aloggrep alpha_4_1_2022_____18_33_46.alog WFLAG --v --final‘)
$ echo $FOO
8
```

```
$ FOO=(‘aloggrep alpha_10_5_2022_____14_31_06.alog MISSION_HASH --v --first‘)
$ echo $MISSION_HASH
2211-0528V-SOUR-CLUB
```

6 The alogrm Tool

The alogrm tool will prune a given .alog file by removing lines of the original file that contain log entries for a user-specified list of MOOS variables or MOOS processes (sources). It may be fairly viewed as the complement of the aloggrep tool.

6.1 Command Line Usage for the alogrm Tool

```
 $ alogrm --help or -h
```

```
 Usage:
  alogrm in.alog [VAR] [SRC] [out.alog] [OPTIONS]

Synopsis:
   Remove the entries matching the given MOOS variables or sources
   from the given .alog file and generate a new .alog file.

Standard Arguments:
  in.alog  - The input logfile.
  out.alog - The newly generated output logfile. If no
             file provided, output goes to stdout.
  VAR      - The name of a MOOS variable
  SRC      - The name of a MOOS process (source)

Options:
  -h,--help     Displays this help message
  -v,--version  Displays the current release version
  -f,--force    Force overwrite of existing file
  -q,--quiet    Verbose report suppressed at conclusion
  --nostr       Remove lines with string data values
  --nonum       Remove lines with double data values
  --clean       Remove lines that have a timestamp that is
                non-numerical or lines w/ no 4th column

Further Notes:
  (1) The second alog is the output file. Otherwise the
      order of arguments is irrelevent.
  (2) VAR* matches any MOOS variable starting with VAR
  (3) See also: alogscan, aloggrep, alogclip, alogview
```

Note that, in specifying items to be filtered out, there is no distinction made on the command line that a given item refers to a entry's variable name or an entry's source, i.e., MOOS process name.

6.2 Example Output from the alogrm Tool

The output shown below was generated from the `alpha.alog` file generated by the Alpha example mission.

```
$ alogrm alpha.alog NAV_* new.alog
```

```
Processing on file : alpha.alog
  Total lines retained: 47396 (46.70%)
  Total lines excluded: 54099 (53.30%)
  Total chars retained: 6453494 (66.21%)
  Total chars excluded: 3293774 (33.79%)
  Variables retained: (48) BHV_IPF, CREATE_CPU, CYCLE_INDEX, DB_CLIENTS,
DB_TIME, DB_UPTIME, DEPLOY, DESIRED_HEADING, DESIRED_RUDDER, DESIRED_SPEED,
DESIRED_THRUST, HELM_IPF_COUNT, HELM_MAP_CLEAR, HSLINE, USIMMARINE_STATUS,
IVPHELM_DOMAIN, IVPHELM_ENGAGED, IVPHELM_MODESET, IVPHELM_POSTINGS,
IVPHELM_STATEVARS, IVPHELM_SUMMARY, LOGGER_DIRECTORY, LOOP_CPU, MOOS_DEBUG,
MOOS_MANUAL_OVERIDE, NODE_REPORT_LOCAL, PC_hsline, PC_waypt_return,
PC_waypt_survey, PHELMIVP_STATUS, PLOGGER_CMD, PLOGGER_STATUS,
PMARINEPID_STATUS, PMARINEVIEWER_STATUS, PNODEREPORTER_STATUS,
PWT_BHV_HSLINE, PWT_BHV_WAYPT_RETURN, PWT_BHV_WAYPT_SURVEY, RETURN,
STATE_BHV_HSLINE, STATE_BHV_WAYPT_RETURN, STATE_BHV_WAYPT_SURVEY,
SURVEY_INDEX, SURVEY_STATUS, VIEW_POINT, VIEW_SEGLIST, WPT_INDEX, WPT_STAT
```

7 The aloghelm Tool

The `aloghelm` tool provides a few handy ways of looking at helm activity over the course of a given single alog file. This includes:

- *Life Events*: Using the `--life/-l` option, every spawning or death of a behavior is sorted into a list of life events. Section 7.1.
- *Mode Changes*: Using the `--modes/-m` option, every helm mode change is sorted into a list of chronological entries. Section 7.2.
- *Behavior States*: Using the `--bhvs/-b` option, every instance where a behavior changes states is recorded and sorted into a list of chronological entries. Section 7.3.

In each mode, the user may additionally specify one or more MOOS variables to be interleaved in the report as they occur chronologically

7.1 The Life Events (–life) Option in the aloghelm Tool

The *life events* option in `aloghelm` will scan the given alog file for all life events, defined by the spawning or destruction of a behavior instance. This information is posted by the helm in the `IVPHELM_LIFE_EVENT` variable. Example output is show below:

```
$ aloghelm file.alog --life
```

```
Processing on file : henry.alog
++++++++++ (100,000) lines
++++++++++ (200,000) lines
+++
233,736 lines total.
10 life events.


      ************************************************
      *        Summary of Behavior Life Events          *
      ************************************************

Time    Iter  Event  Behavior      Behavior Type       Spawning Seed
------  ----  -----  ------------  ------------------  ------------------------------
 41.27     1  spawn  loiter        BHV_Loiter          helm_startup
 41.27     1  spawn  waypt_return  BHV_Waypoint        helm_startup
 41.27     1  spawn  station-keep  BHV_StationKeep     helm_startup
316.78   995  spawn  ac_avd_gilda  BHV_AvoidCollision  name=avd_gilda # contact=gilda
369.72  1191  death  ac_avd_gilda  BHV_AvoidCollision
482.92  1601  spawn  ac_avd_gilda  BHV_AvoidCollision  name=avd_gilda # contact=gilda
545.18  1833  death  ac_avd_gilda  BHV_AvoidCollision
654.70  2228  spawn  ac_avd_gilda  BHV_AvoidCollision  name=avd_gilda # contact=gilda
751.87  2591  death  ac_avd_gilda  BHV_AvoidCollision
809.85  2799  spawn  ac_avd_gilda  BHV_AvoidCollision  name=avd_gilda # contact=gilda
```

The actual output, by default, is color-code green for all spawnings and black for all deaths. The color-coding can be turned off with the additional command line argument `--nocolor`.

7.2 The Modes (–modes) Option in the aloghelm Tool

The *modes* option in `aloghelm` will scan the given alog and report all instances of a helm mode change.

```
$ aloghelm file.alog --modes
```

```
Processing on file : /Users/mikerb/henry.alog
================================================
45.221 Mode: ACTIVE:LOITERING
================================================
92.687 Mode: ACTIVE:STATION-KEEPING
================================================
120.919 Mode: ACTIVE:LOITERING
================================================
386.632 Mode: ACTIVE:RETURNING
================================================
413.980 Mode: ACTIVE:LOITERING
================================================
558.254 Mode: ACTIVE:RETURNING
================================================
584.283 Mode: ACTIVE:LOITERING
================================================
663.162 Mode: ACTIVE:STATION-KEEPING
================================================
703.517 Mode: ACTIVE:LOITERING
================================================
766.938 Mode: ACTIVE:RETURNING

233,736 lines total.
```

Using the `--mode` option, it is sometimes helpful to augment the output to include certain other variable postings, by simply naming the MOOS variable on the command line. The variables and mode changes will be presented on the screen in their chronological order. For example:

```
$ aloghelm file.alog --modes CONTACT_RESOLVED
```

```
Processing on file : /Users/mikerb/henry.alog
================================================
45.221 Mode: ACTIVE:LOITERING
================================================
92.687 Mode: ACTIVE:STATION-KEEPING
================================================
120.919 Mode: ACTIVE:LOITERING
373.392         CONTACT_RESOLVED     pHelmIvP:1190:ac_avd_gilda GILDA
================================================
386.632 Mode: ACTIVE:RETURNING
================================================
413.980 Mode: ACTIVE:LOITERING
548.838         CONTACT_RESOLVED     pHelmIvP:1832:ac_avd_gilda GILDA
================================================
558.254 Mode: ACTIVE:RETURNING
================================================
584.283 Mode: ACTIVE:LOITERING
================================================
663.162 Mode: ACTIVE:STATION-KEEPING
================================================
703.517 Mode: ACTIVE:LOITERING
755.509         CONTACT_RESOLVED     pHelmIvP:2590:ac_avd_gilda GILDA
================================================
766.938 Mode: ACTIVE:RETURNING

233,736 lines total.
```

7.3 The Behaviors Option in the aloghelm Tool

The *behaviors* option in `aloghelm` will scan the given alog file taking note of all helm iterations where there is a change to one or more of the four groups of (a) active, (b) running, (c) idle, or (d) completed behaviors. Example output is show below:

```
$ aloghelm file.alog --bhvs
```

```
Processing on file : henry.alog
================================================
45.221 Mode: ACTIVE:LOITERING
 - - - - - - - - - - - - - - - - - - - - - - - - -
45.225    (1) Active:    loiter
45.225    (1) Running:
45.225    (1) Idle:      waypt_return,station-keep
================================================
92.687 Mode: ACTIVE:STATION-KEEPING
 - - - - - - - - - - - - - - - - - - - - - - - - -
92.689    (172) Active:    station-keep
92.689    (172) Running:
92.689    (172) Idle:      loiter,waypt_return
================================================
120.919 Mode: ACTIVE:LOITERING
 - - - - - - - - - - - - - - - - - - - - - - - - -
120.921    (274) Active:    loiter
120.921    (274) Running:
120.921    (274) Idle:      waypt_return,station-keep
 - - - - - - - - - - - - - - - - - - - - - - - - -
320.786    (995) Active:    loiter,ac_avd_gilda
320.786    (995) Running:
320.786    (995) Idle:      waypt_return,station-keep
 - - - - - - - - - - - - - - - - - - - - - - - - -
345.778    (1090) Active:    loiter
345.778    (1090) Running:   ac_avd_gilda
345.778    (1090) Idle:      waypt_return,station-keep
 - - - - - - - - - - - - - - - - - - - - - - - - -
373.642    (1191) Active:    loiter
373.642    (1191) Running:
373.642    (1191) Idle:      waypt_return,station-keep
373.642    (1191) Completed: ac_avd_gilda
================================================
386.632 Mode: ACTIVE:RETURNING
 - - - - - - - - - - - - - - - - - - - - - - - - -
386.636    (1238) Active:    waypt_return
386.636    (1238) Running:
386.636    (1238) Idle:      loiter,station-keep
386.636    (1238) Completed: ac_avd_gilda
================================================
```

In some cases, there is interest in a particular behavior in the this kind of output. To make it easier to visually parse, the `--watch=BHV` option can be used to draw attention to each the particular behavior changes state. Example output is shown below. The primary difference is the `CHANGE` tag for each instance of a state change. In the terminal, such lines are also rendered in a different color.

```
 $ aloghelm file.alog --bhvs --watch=loiter
```

```
Processing on file : henry.alog
================================================
45.221 Mode: ACTIVE:LOITERING
 - - - - - - - - - - - - - - - - - - - - - - - - -
45.225    (1) Active:    loiter             CHANGE
45.225    (1) Running:
45.225    (1) Idle:      waypt_return,station-keep
================================================
92.687 Mode: ACTIVE:STATION-KEEPING
 - - - - - - - - - - - - - - - - - - - - - - - - -
92.689    (172) Active:    station-keep
92.689    (172) Running:
92.689    (172) Idle:      loiter,waypt_return            CHANGE
================================================
120.919 Mode: ACTIVE:LOITERING
 - - - - - - - - - - - - - - - - - - - - - - - - -
120.921    (274) Active:    loiter             CHANGE
120.921    (274) Running:
120.921    (274) Idle:      waypt_return,station-keep
 - - - - - - - - - - - - - - - - - - - - - - - - -
320.786    (995) Active:    loiter,ac_avd_gilda
320.786    (995) Running:
320.786    (995) Idle:      waypt_return,station-keep
 - - - - - - - - - - - - - - - - - - - - - - - - -
345.778    (1090) Active:    loiter
345.778    (1090) Running:   ac_avd_gilda
345.778    (1090) Idle:      waypt_return,station-keep
 - - - - - - - - - - - - - - - - - - - - - - - - -
373.642    (1191) Active:    loiter
373.642    (1191) Running:
373.642    (1191) Idle:      waypt_return,station-keep
373.642    (1191) Completed: ac_avd_gilda
================================================
386.632 Mode: ACTIVE:RETURNING
 - - - - - - - - - - - - - - - - - - - - - - - - -
386.636    (1238) Active:    waypt_return
386.636    (1238) Running:
386.636    (1238) Idle:      loiter,station-keep            CHANGE
386.636    (1238) Completed: ac_avd_gilda
================================================
```

7.4 Command Line Usage for the aloghelm Tool

```
$ aloghelm --help or -h
```

Listing 7.1: Command line usage for the aloghelm *tool.*

```
1  Usage:
2    aloghelm file.alog [OPTIONS] [MOOSVARS]
3
4  Synopsis:
5    Perform one of several optional helm reports based on
6    helm output logged in the given .alog file.
7
8  Options:
```

```
 9    -h,--help     Displays this help message
10    -v,--version  Displays the current release version
11    -l,--life     Show report on IvP Helm Life Events
12    -b,--bhvs     Show helm behavior state changes
13    -m,--modes    Show helm mode changes
14    --watch=bhv   Watch a particular behavior for state change
15    --nocolor     Turn off use of color coding
16    --notrunc     Don't truncate MOOSVAR output (on by default)
17
18  Further Notes:
19    (1) The order of arguments is irrelevent.
20    (2) Only the first specified .alog file is reported on.
21    (3) Arguments that are not one of the above options or an
22        alog file, are interpreted as MOOS variables on which
23        to report as encountered.
```

8 The alogiter Tool

The alogiter tool will analyze the ITER_GAP and ITER_LEN information produced by any appcasting MOOS app. These variables indicate the ability of an application to keep up with the requested apptick frequency. For example PHELMIVP_ITER_GAP will be close to 1.0 when configured with an apptick of 4, and the observed apptick is also 4. The gap value will be around 2 if the observed apptick is around 2. The PHELMIVP_ITER_LEN is the elapsed time between the start and end of the helm iterate loop.

8.1 Command Line Usage for the alogiter Tool

```
$ alogiter --help or -h
```

Listing 8.2: Command line usage for the alogiter tool.

```
 1  $ alogrm -h
 2
 3  Usage:
 4    alogiter in.alog [OPTIONS]
 5
 6  Synopsis:
 7    Analyze the ITER_GAP and ITER_LEN information provided by
 8    all applications recorded in the given alog file.
 9
10  Standard Arguments:
11    file.alog - The input logfile.
12
13  Options:
14    -h,--help     Displays this help message
15    -v,--version  Displays the current release version
16
17  Further Notes:
18    See also: alogscan, alogrm, alogclip, alogview, aloggrep
```

8.2 Example Output from the alogiter Tool

The output shown in Listing 3 was generated from the alpha.alog file generated by the Alpha example mission, at time warp 20.

Listing 8.3: Example alogiter output applied to the alpha.alog file.

```
$ alogiter alpha.alog

Processing on file : MOOSLog_22_4_2015_____13_25_19.alog
                 GAP     GAP       PCT     PCT     PCT
AppName          MAX     AVG       >1.25   >1.50   >2.0
-------------  -----   -----      -----   -----   -----
PHELMIVP         1.26    1.11      0.005   0.000   0.000
PMARINEVIEWER    1.10    1.07      0.000   0.000   0.000
PNODEREPORTER    1.25    1.15      0.008   0.000   0.000
UPROCESSWATCH    1.26    1.12      0.014   0.000   0.000
USIMMARINE       1.27    1.12      0.009   0.000   0.000

                 LEN     LEN       PCT     PCT     PCT     PCT
AppName          MAX     AVG       >0.25   >0.50   >0.75   >1.0
-------------  -----   -----      -----   -----   -----   -----
PHELMIVP         0.08    0.04      0.000   0.000   0.000   0.000
PMARINEVIEWER    0.00    0.00      0.000   0.000   0.000   0.000
PNODEREPORTER    0.01    0.00      0.000   0.000   0.000   0.000
UPROCESSWATCH    0.01    0.00      0.000   0.000   0.000   0.000
USIMMARINE       0.02    0.00      0.000   0.000   0.000   0.000

Mission Summmary
--------------------------
 Collective APP_GAP: 1.11
 Collective APP_LEN: 0.01
```

9 The alogsplit Tool

The alogsplit tool will split a given .alog file into a directory containing a file for each MOOS variable found in the .alog file. This is essentially the first stage of pre-processing done at the outset of launching the alogview tool. It is implement here as a stand-alone app to be used for purposes other than alogview. It may also be useful as a command-line tool for preparing multiple .alog files from a shell script well before the first time they are used in alogview.

This is a new tool in Release 15.4 coinciding with the major re-write of the alogview tool also released in 15.4.

9.1 Naming and Cleaning the Auto-Generated Split Directories

The name of the *split directory* created by alogsplit is determined automatically from the .alog filename. For a file name alpha.alog, the directory created will be alpha_alvtmp/ by default. This can be overridden with the command line switch --dir=my_dirname. The fairly distinctive _alvtmp suffix was chosen to facilitate cleaning these auto-generated temporary directories with a simple shell script, alv_rm:

```
#!/bin/bash
find . -name '*_alvtmp'  -print -exec rm -rfv {} \;
```

The above script is found in the moos-ivp/bin directory and will remove (without prompting for confirmation) all split directories in the current directory and sub-directories.

9.2 Command Line Usage for the alogsplit Tool

```
$ alogsplit --help or -h
```

Listing 9.4: Command line usage for the `alogsplit` *tool.*

```
$ alogsplit -h

Usage:
  alogsplit in.alog [OPTIONS]

Synopsis:
  Split the given alog file into a directory, within which
  each MOOS variable is split into it's own (klog) file
  containing only that variable. The split will also create
  a summary.klog file with summary information.

  Given file.alog, file_alvtmp/ directory will be created.
  Will not overwrite directory if previously created.
  This is essentially the operation done at the outset of
  launching the alogview applicaton.

Standard Arguments:
  in.alog  - The input logfile.

Options:
  -h,--help      Displays this help message
  -v,--version   Displays the current release version
  --verbose      Show output for successful operation
  --dir=DIR      Override the default dir with given dir.
```

9.3 Example Output from the alogsplit Tool

The output shown in Listing 5 was generated from the `alpha.alog` file generated by the Alpha example mission.

Listing 9.5: Example `alogsplit` *directory applied to the* `alpha.alog` *file.*

```
$ alogsplit alpha.alog

APPCAST.klog                  IVPHELM_CPU.klog              NODE_REPORT_LOCAL.klog
APPCAST_REQ.klog              IVPHELM_CREATE_CPU.klog       PHELMIVP_ITER_GAP.klog
APPCAST_REQ_ALL.klog          IVPHELM_DOMAIN.klog           PHELMIVP_ITER_LEN.klog
APPCAST_REQ_ALPHA.klog        IVPHELM_IPF_CNT.klog          PLOGGER_CMD.klog
BHV_IPF_waypt_return.klog     IVPHELM_ITER.klog             PMARINEVIEWER_ITER_GAP.klog
BHV_IPF_waypt_survey.klog     IVPHELM_LIFE_EVENT.klog       PMARINEVIEWER_ITER_LEN.klog
CYCLE_INDEX.klog              IVPHELM_LOOP_CPU.klog         PMV_CONNECT.klog
CYCLE_INDEX_SURVEYING.klog    IVPHELM_MODESET.klog          PNODEREPORTER_ITER_GAP.klog
DB_CLIENTS.klog               IVPHELM_REGISTER.klog         PNODEREPORTER_ITER_LEN.klog
DB_EVENT.klog                 IVPHELM_STATE.klog            PROC_WATCH_EVENT.klog
DB_QOS.klog                   IVPHELM_STATEVARS.klog        PROC_WATCH_FULL_SUMMARY.klog
DB_RWSUMMARY.klog             IVPHELM_SUMMARY.klog          PROC_WATCH_SUMMARY.klog
DB_TIME.klog                  LOGGER_DIRECTORY.klog         PROC_WATCH_TIME_WARP.klog
DB_UPTIME.klog                MOOS_DEBUG.klog               RETURN.klog
DEPLOY.klog                   MOOS_MANUAL_OVERRIDE.klog     SIMULATION_MODE.klog
DESIRED_HEADING.klog          NAV_DEPTH.klog                TRUE_X.klog
DESIRED_RUDDER.klog           NAV_HEADING.klog              TRUE_Y.klog
DESIRED_SPEED.klog            NAV_HEADING_OVER_GROUND.klog  UPROCESSWATCH_ITER_GAP.klog
DESIRED_THRUST.klog           NAV_LAT.klog                  UPROCESSWATCH_ITER_LEN.klog
HELM_MAP_CLEAR.klog           NAV_LONG.klog                 USIMMARINE_ITER_GAP.klog
IVPHELM_ALLSTOP.klog          NAV_PITCH.klog                USIMMARINE_ITER_LEN.klog
```

```
24  IVPHELM_ALLSTOP_DEBUG.klog    NAV_SPEED.klog                USM_DRIFT_SUMMARY.klog
25  IVPHELM_BHV_ACTIVE.klog       NAV_SPEED_OVER_GROUND.klog    USM_FSUMMARY.klog
26  IVPHELM_BHV_CNT.klog          NAV_X.klog                    VISUALS.klog
27  IVPHELM_BHV_CNT_EVER.klog     NAV_Y.klog                    summary.klog
28  IVPHELM_BHV_IDLE.klog         NAV_YAW.klog
29  IVPHELM_BHV_RUNNING.klog      NAV_Z.klog
```

Notice the `summary.klot` file on line 27. It contains some meta information gathered during the split process that is useful for `alogview` in fetching information at run time.

10 The alogpare Tool

The `alogpare` tool is a utility for pruning alog files by removing certain alog entries outside certain time windows. The time windows are defined by a user-defined time duration around the entries of further user-defined variables in the log file. The idea is that some robot missions have events of interest, e.g., a near collision event, where retaining all data just before and after the event is critical to analyzing what may have gone wrong. Perhaps certain high data rate log entries outside these critical event windows may be removed without any loss in utility to the users. In some cases this reduction in logged data may dramatically ease the acrhiving of these log files.

This was a new tool in Release 17.7 but was not documented until the following release.

10.1 Mark Variables Define Events of Interest

A *mark variable* is a MOOS variable provided to `alogpare` on the command line to indicate an event of interest. From the perspective of `alogpare`, the value of the mark variable does not matter. One or variables may be provided. For example:

```
$ alogpare --markvars=ENCOUNTER,NEAR_MISS
```

The `alogpare` utility will make an initial pass through the alog file and make not of each instance of a mark variable. A window of time, given by the command line parameter `--pare_window`, will will be associated around each instance of a mark variable. If windows overlap, that's fine. The during of the pare window is 30 seconds by default, even split in time before and after the mark event. This may be adjusted on the command line. For example:

```
$ alogpare --markvars=ENCOUNTER --pare_window=60
```

The `alogpare` utility will make a second pass through the alog file pruning log entries *outside* the pare windows, based on variables on the pare list.

10.2 The Pare List of Variables to be Pared

Variables on the *pare list* indicate which lines of an alog file are to be removed, outside of pare windows. The pare list is defined on the command line with:

```
$ alogpare --markvars=ENCOUNTER --pare_window=60 --parevars=BHV_IPF,BIG_ENTRY
```

Typically these variables constitute relatively large portions of an alog file, and provide little value outside the pare windows.

10.3 The Hit List of Variables to be Removed Completely

The alogpare utility also provides the means for removing named variables outright, regardless of where they occur relative to a pare window. These variables are on the *hit list*. For example:

```
$ alogpare --varkvars=ENCOUNTER --parevars=BHV_IPF --hitvars=ITER_GAP
```

This functionality is also achieved with the alogrm utility, and is provided in this tool just as a convenience.

10.4 Command Line Usage for the alogpare Tool

```
$ alogpare --help or -h
```

Listing 10.6: Command line usage for the alogpare tool.

```
$ alogpare -h

  Usage:
  alogpare .alog [out.alog] [OPTIONS]

Synopsis:
  Pare back the given alog file in a two-pass manner.
  First pass detects events defined by given mark vars.
  The second pass removes lines with vars on the pare
  list if they are not within pare_window seconds of
  an event line. It also removes lines with vars on the
  hitlist unconditionally. Latter could also be done
  with alogrm.
  The original alog file is not altered.

Options:
  -h,--help         Displays this help message
  -v,--version      Display current release version
  --verbose         Enable verbose output
  --markvars=<L>    Comma-separated list of mark vars
  --hitvars=<L>     Comma-separated list of hit vars
  --parevars=<L>    Comma-separated list of pare vars
  --pare_window=<N> Set window to N seconds (default 30)

Examples:
  alogpare --markvars=ENCOUNTER --parevars=BHV_IPF
           original.alog smaller.alog
  alogpare --markvars=ENCOUNTER
           --parevars=BHV_IPF,VIEW_*
           --hitvars=*ITER_GAP,*ITER_LEN,DB_QOS
           --pare_window=10
           original.alog smaller.alog

Further Notes:
  (1) The order of alogfile args IS significant.
  (2) The order of non alogfile args is not significant.
```

10.5 Planned additions to the alogpare Utility

- Soft parevars: removing perhaps every other entry outside pare window. Or remove success entries with identical values outside the pare window.
- Separate specification for pare_window time. Currently the window is split evenly around the mark event. Some user may want more control.
- Pattern matching: Add support for specifying sets of variables with simple wildcard prefix or suffix, e.g., `NAV_*` or `*_REPORT`.

11 The alogcd Tool

The `alogcd` tool is a utility for scanning a given alog file and tallying the number of encounters, near misses, and collisions. This utility works under the assumption that another utility had been running during the mission, and monitoring encounters, near misses and collisions. It assumes that these three events were separately noted with MOOS variables that also indicate the closest point of approach (CPA) range for each event. And it also assumes that these three variables were logged in the alog file.

The `alogcd` utility uses the MOOS variables `ENCOUNTER`, `NEAR_MISS`, and `COLLISION`. For now, these three variables are hard-coded in this utility. The `uFldCollisionDetect` utility is one utility capable of generating this kind of output. If there is collision to report, the report will also show the CPA value for the worst collision encounter.

An example run may produce output similar to:

```
$ alogcd file.alog

7,686 total alog file lines.

========================================
Collision Report:
========================================
Encounters:  27  (avg 16.93 m)
Near Misses: 6  (avg 10.18 m)
Collisions:  3  (avg 5.55 m)
Collision Worst: 3.87
```

11.1 Producing a Time-Stamped file of Collisions and Near Misses

The near misses and collisions are the real events of interests, and if the standard summary report is not enough, a time stamped list of each near miss and collision may be written to a file, if the `--tfile=filename` parameter is provided. For example, the six near misses and three collisions reported above could be written to file with:

```
$ alogcd file.alog --tfile=myfile
$ cat myfile
69.149,COLLISION,5.17
231.826,NEAR_MISS,9.41
351.374,NEAR_MISS,10.33
556.815,NEAR_MISS,10.35
592.976,NEAR_MISS,9.69
792.884,COLLISION,3.87
1065.484,NEAR_MISS,11.51
1129.862,COLLISION,7.61
1275.018,NEAR_MISS,9.82
```

The first column is the timestamp from the alog file, the second column is the type of encounter (near miss or collision), and the third column is the CPA distance for that encounter.

11.2 The Terse Output Option

For a super terse, one line report, use the following, which produces the below output for the same alog file as in the example above:

```
$ alogcd file.alog
27/6/3
```

27 encounters, 6 near misses, 3 collisions.

11.3 Command Line Return Values

The ultimate terse output is none at all! In this case we're only interested in the return value of alogcd. This can be used for example in a shell script to launch a series of simulations, altering the configuration parameters until no collisions are detected. The following (integer) return values are implemented:

- [0]: The alog file was found and readable, encounters were indeed reported, and no collisions were reported. The success condition.
- [1]: The alog file was not found or it was not readable.
- [2]: The alog file was indeed found and was readable, but sadly, collisions were reported.
- [3]: The alog file was indeed found and was readable, and no collisions were reported, but no encounters were reported either. Something is amiss. Either the vehicles never even got close enough to each other to constitute an encounter, or a monitoring app like uFldCollisionDetect was not even running.

11.4 Command Line Usage for the alogcd Tool

```
$ alogcd --help or -h
```

Listing 11.7: Command line usage for the alogcd tool.

```
1  $ alogcd -h
```

```
 2
 3  Usage:
 4    alogcd .alog [OPTIONS]
 5
 6  Synopsis:
 7    Scan an alog file for collision detection reports.
 8    Tally the totals and averages, and optionally create
 9    a file holding all the timestamps of events.
10
11    By default, it scans for events defined by postings
12    to the following three MOOS variables:
13
14    (1) COLLISION
15    (2) NEAR_MISS
16    (3) ENCOUNTER
17
18  Options:
19    -h,--help       Displays this help message
20    -v,--version    Display current release version
21    -t,--terse      Write terse output.
22
23  Returns:
24    0 if alog file ok, encounters detected, no collisions.
25    1 if alog file not ok, unable to open.
26    2 if alog ok, but collisions were detected
27    3 if alog ok, no collisions or encounters detected
```

11.5 Planned additions to the alogcd Utility

- Allow the key MOOS variables to be provided as parameters, rather then fixed to `ENCOUNTER`, `NEAR_MISS`, and `COLLISION`.
- Support cmd line option like `--collision_count` which produces the integer value of collision counts as the command line return value. Perhaps the same for `--near_miss_count` or `encounter_count`.

12 The alogcat Tool

A concise form of this documentation is available with `alogcat --help, -h`, and the web version of this content can be quickly opened with `alogcat --web, -w`.

The `alogcat` tool is a utility for concatenating a given set of alog files into a new single alog file. Recall that each alog file has a header at the beginning of the file with meta information. This includes the starting timestamp which allows all further timestamps to be relative to the starting time. So to concatenate alog files, we cannot simply just append one file onto the end another. If that were done, there would be multiple header blocks in the file and different blocks of data with timestamps relative to different start times.

Why would we need this tool? In certain field exercises, occasionally an operator may decide to stop the mission (killing the MOOS community), and restart with perhaps a slight modification in an important parameter. In such cases, two sets of log files will be produced, one from before the restart and one from after. The two of them may constitute a valid mission log file, but they are now split into two. The `alogcat` utility can be used for merging them back into one.

The alogcat tool performs a proper concatenation. First, it determines the chronological ordering of the provided alog files. It will use the header block and starting time of the earliest file. For the remaning files, (a) the relative time to the first file is calculated, (b) the header block of the later file(s) is removed, (c) the log entries of the later file(s) are appended to the end of the earlier file with timestamps appropriately adjusted along the way.

As an example, consider two alog files created from the same mission, `fil1.alog` and `file2.alog`. They can be joined with:

```
$ alogcatfile1.alog file2.alog --new=final.alog --verbose
Performing a pre-check on the list of alog files...
Processing file: file1.alog
Processing file: file2.alog
Created new alog file: final.alog
```

Overlapping files will produce an error and no new file will be created. If a second log file is created by re-starting the logger after it has been stopped during a mission, it is possible that initial postings will have negative time stamps. This is due to the MOOSDB delivering the mail with the latest values. These postings may have occurred before the logger re-started, and in fact may represent duplicate postings found in the earlier log file. These postings are ignore. Keep in mind that any posting in subsequent log files that have a negative timestamp will be ignored with the files are concatenated.

If the new file to be created already exists, no action will be taken. This can be overridden with the `--force, -f` parameter.

13 The alogavg Tool

The post-mission analysis pipeline starts with raw alog files and ends with plotted data. The raw data in alog files cannot be directly plotted without some filtering and analysis steps. The alogavg tool is one of the tools for performing these steps. These steps could also be done within the plotting language such as Matlab or Python, but our preference is to have general tools in the ALog Toolbox for these steps to allow users to choose between plotting tools without having to re-write the data preparation steps in each plotting language.

A concise form of this documentation is available on the command line:

```
$ alogavg --help    (or -h)
```

The web version of this content can be quickly summoned from the command line with:

```
$ alogavg --web    (or -w)
```

13.1 Input Format for alogavg

The alogavg tool ingests a single file, using the `--file=file.txt` argument. This file will have two columns of data, the domain (x) in the first column, and the range (y) in the second column. A file

of this type may have been produced by alogrep directly from the raw alog file. No ordering is presumed. A simple example:

```
2  289.09
1  357.331
2  287.762
1  358.505
2  289.645
1  358.952
```

For now it is assumed that the column separator is white space (ASCII 32). White space at either the beginning or end of each line is ignored, and it does not matter how much white space is between the two columns.

13.2 Output Format for alogavg

The output of alogavg will be five columns:

- The domain (x values), ordered
- The average of the range (y values)
- The minimum range value for the given domain value
- The maximum range value for the given domain value
- The standard deviation of range values for the given domain value

A simple example, using the data above, stored in `file.log`:

```
$ alogavg file.log
1 358.262667 357.331 358.952 0.683596
2 288.832333 287.762 289.645 0.790028
```

By default the columns of the output are padded to line up all columns in a more human readable format. With the `--noformat` option, the columns will all be separated by a single white space.

14 The alogmhash Tool

The alogmhash command-line tool will examine a given single alog file and discern key pieces of information related to the mission hash. It's primary intended use is to be invoked within a script as step in the automated archiving of log files. But it can also be run on the command line to provide some useful information about the mission hash of an alog file:

```
$ alogmhash LOG_ABE_26_6_2023_____22_00_52.alog
Analyzing odometry in file : LOG_ABE_26_6_2023_____22_00_52.alog

6,697 total alog file lines.
-----------------------------------
MHash:     230626-2200N-MALT-PAIR
Odometry:  0
Duration:  73.2
Node Name: abe
UTC:       1687831252.73
Full List of MHashes noted:
 [230626-2200N-MALT-PAIR]: 0.00 secs
```

The condensed output version of this tool is the primary use case. It will produce a slingle line of ouput that will be saved in an `.mhash` file. This information is critical to the archiving pipeline

```
$ alogmhash --all LOG_ABE_26_6_2023_____22_00_52.alog
mhash=230626-2200N-MALT-PAIR,odo=0.0,duration=73.2,name=abe,utc=1687831252,xhash=ABE-073S-000M
```

14.1 Output Components

The mhash utility will scour the log file for the following pieces of information.

- The mission hash (mhash)
- The UTC start time of the mission
- The odometry, e.g., distance travelled
- The duration of the log file
- The node name

The mission hash is obtained by examining the alog file for entries to the variable MISSION_HASH. An example is:

mhash=230626-2200N-MALT-PAIR,utc=1687831252.73

If there is only one value for this variable in the alog file, then identifying the mission hash and the UTC start time is pretty easy. However, if the alog file is from a vehicle, it's possible that the shoreside mission may have been restarted one or more times in the duration of the vehicle log file. This would result in the vehicle log file having multiple values for the MISSION_HASH. For this reason alogmhash tallies the odometry distance accumulated while each distinct MISSION_HASH is the prevailing value. In the end, the one with the longest odometry distance is the one chosen. The assumption is that, when or if a shoreside mission is re-started, it is usually while the vehicle is sitting in a pre-launch mode, in which case the mission hash values from earlier starts should be disregarded.

The odometry distance calculated by alogmhash is derived simply from the logged NAV_X and NAV_Y values.

The duration value calculated by alogmhash is derived simply from timestamp of the last logged entry in the alog file against the UTC start time recorded at the top of the alog file.

The node name is either the vehicle name or "shoreside". It is derived by examining a publication to the MOOS variable DB_UPTIME and noting the source (MOOS App) name that published it, which is always the MOOSDB with the format MOOSDB_node. The component to the right of the underscore is regarded as the node name. The node name of "shoreside" is always shortened to "shore"

14.2 Command Line Usage for the alogmhash Tool

```
$ alogmhash --help or -h
```

Listing 14.8: Command line usage for the alogmhash tool.

```
Usage:
  alogmhash file.alog [OPTIONS]

Synopsis:
  alogmhash will read the one given alog file and
  parse for key values used in a .mhash cache file.
  Value Examples:
    MHASH:    230513-1453B-ICED-OWEN
    ODO:      423.2
    DURATION: 874.3
    NAME:     abe
    UTC:      33678848885.977
  If multiple MISSION_HASH values are present in
  the given log file, the posting value with the
  highest odemetry distance is the one chosen. This
  guards against the situation where a shoreside
  re-starts after the logging has started and thus
  generates a new mission hash. Rule of thumb is to
  regard the MISSION_HASH that prevailed for the
  highest odometry as the one true MISSION_HASH.

Options:
  -h,--help       Displays this help message
  -v,--version    Display current release version
  -t,--terse      terse output (No newline/CRLF)

  -m,--mhash      Report MHASH w/ longest odometry
  -o,--odo        Report total odometry
  -d,--duration   Report total duration
  -n,--name       Report node/vehicle name
  -u,--utc        Report UTC start time
  -a,--all        Report mhash,odo,dur,name

  --web,-w
    Open browser to:
    https://oceanai.mit.edu/ivpman/apps/alogmhash

Returns:
  0 if alog file ok.
  1 if alog file not ok, unable to open.

Further Notes:
  (1) The order of arguments is irrelevent.
  (2) Only last given .alog file is reported on.
  (3) Intended to be used in mhash_tag.sh script.

(1) Example:
  $ alogmhash *.alog --all
    mhash=230512-2147I-ICED-OWEN,odo=1507.5,     \
```

```
   duration=451.3,name=alpha,utc=33678848885.977
(2) Example:
  $ alogmhash *.alog --mhash
   mhash=230512-2147I-ICED-OWEN
```

15 The alogeval Tool

The `alogeval` tool will scan a given alog file and apply test criteria found in a separate input file. The test criteria should be similar to that used by `pMissionEval`, specifying criteria for *when* the test should be applied, and then the criteria for passing or failing the test. The advantage of `alogeval` over `pMissionEval` is that the user can repeatedly adjust the evaluation criteria and re-run the evaluations. With `pMissionEval`, the test criteria loaded at run-time will produce mission evaluations that are based on the run-time criteria only.

A concise form of this documentation is available on the command line:

```
$ alogeval --help    (or -h)
```

The web version of this content can be quickly summoned from the command line with:

```
$ alogeval --web    (or -w)
```

15.1 Test Criteria Input File

The test criteria is provided in a separate input file. If the file has the suffix `.txt`, it can simply be provided, otherwise use the `--testfile=filename` argument. The following two are equivalent:

```
$ alogeval mission.alog criteria.txt
$ alogeval mission.alog --testfile=criteria.txt
```

15.2 Test Criteria Logic Test Sequence

The primary configuration structure is a *logic test sequence*, comprised of one or more *logic tests*. The sequence of tests may be ordered by time, but may also be time invariant. The overall test evaluation is completed when either (a) all logic tests in the sequence have been evaluated and passed, or (b) when any one of the logic tests in the sequence has failed.

A logic single test is comprised of:

- lead conditions
- pass conditions
- fail conditions

One or more lead conditions may be provided. As soon as all lead conditions have been satisfied, the pass/fail conditions are evaluated. If one or more pass conditions have been provided, then

all must satisfied or else the test fails. If one or more fail conditions are provided, then if *any* fail condition is satisfied, the test fails.

A test sequence is comprised one or more tests. Each test has its own conditions. In this case *the order of lines in the configuration file is important* as it determines the grouping. For example, in the Alpha End Flag mission, the sole test is whether the end flag has been set after the fifth waypoint has been hit. This does not test for the possible error case where the endflag is posted after the third or fourth waypoint has been hit. The single logic test could be replace with the following test sequence:

```
lead_condition = WC_FLAG = 3
pass_condition = EFLAG = 0

lead_condition = WC_FLAG = 4
pass_condition = EFLAG = 0
```

Note in this case there are three tests, all tests need to pass if the `pass_flag` is to be posted. Note also that the test `EFLAG=0` will not pass unless it is explicitly published with this value. And the condition `EFLAG!=1` will also not pass unless `EFLAG` has been published. In this example, in the Alpha End Flag mission, the helm initializes the `EFLAG` variable to 0 upon helm launch.

The order of the tasks in the configuration does matter. The app at any given moment will address the next element in the sequence. Only when the last element has been addressed will the result, pass, and fail flags be published. Consider what would happen if the first two of the above tests were reversed? In this case the mission would never finish evaluation. Why?

15.3 Test Output and Return Values

If `alogeval` succeeds, with valid alog file, valid criteria file, and passing all logic tests, it returns with a value of 0. Other possible values:

- 0: Success
- 1: All valid files, but test criteria failed
- 2: Unhandled command line argument
- 3: Missing alog file or could not be opened
- 4: Missing criteria file or could not be opened
- 5: ALog or criteria file could not be parsed

The above return values are returned to the shell on the command line. By default, `alogeval` produces no output to the terminal. Output can be enabled with the `--verbose` flag, for example:

```
$ alogeval file.alog criteria.txt --verbose
BEGIN ALogEvaluator: Handling ALog File:
    14.72909      EFLAG  0   [false,false] (0)
   121.82700    WC_FLAG  1   [false,false] (0)
   162.14515    WC_FLAG  2   [false,false] (0)
   192.43813    WC_FLAG  3   [false,false] (1)
   215.20145    WC_FLAG  4   [true,true]   (2)
EVALUATION COMPLETE
END  ALogEvaluator: Handling ALog File
Result: pass=true
$ echo $?
0
```

In the above output, several lines with timestamps are shown. Variables involved in the criteria logic conditions are noted by `alogeval`. Each time a log entry is encountered with one of these variables, the test criteria is applied, and a line of output is produced. The second column is the MOOS variable name. The third column is the posted value of the MOOS variable. The fourth column contains two Booleans. The first is whether the entire test sequence has been evaluated, and the second Boolean is whether the evaluation passed. If the first is false, the second will false. The fifth column is index of the current logic test. Since there are only two logic tests in this example, with the index is 2, then the evaluations should be complete.

In the line with timestamp at `192` seconds, the `WC_FLAG` posting is 3, which is the first `lead_condition`. Since `EFLAG=0` is the pass condition, this first logic test passes and the logic test index increments from 0 to 1. In the line with timestamp at `215` seconds, the `WC_FLAG` posting is 4, which is the second `lead_condition`. Since `EFLAG=0` is the pass condition, this second logic test passes and the logic test index increments from 0 to 2. Since all logic tests have been evaluated, and all passed, the fourth column on this final line reads `[true,true]`, and the evaluation is complete.

15.4 Examining the Test Sequence Structure

The normal verbose output can be augmented with the `--show_seq` or `-ss` flag. This will show the structure of test criteria before the evaluation, and again after the evaluations. For example:

```
$ alogeval file.alog criteria.txt --verbose -ss
BEGIN ALogEvaluator LogicTestSequence =========
LogicTestSequence: Total Aspects:2
Enabled:   true
Evaluated: false
Satisfied: false
Status:    pending
++++++++++ aspect[0] ++++++++++++
Lead Conditions: (1)  WC_FLAG = 3
Pass Conditions: (1)  EFLAG = 0
Fail Conditions: (none)
Enabled(true), Evaluated(false), Satisfied(false), Status(pending)
++++++++++ aspect[1] ++++++++++++
Lead Conditions: (1)  WC_FLAG = 4
Pass Conditions: (1)  EFLAG = 0
Fail Conditions: (none)
Enabled(true), Evaluated(false), Satisfied(false), Status(pending)
END   ALogEvaluator LogicTestSequence =========

BEGIN ALogEvaluator: Handling ALog File:
    14.72909      EFLAG  0   [false,false] (0)
   121.82700    WC_FLAG  1   [false,false] (0)
   162.14515    WC_FLAG  2   [false,false] (0)
   192.43813    WC_FLAG  3   [false,false] (1)
   215.20145    WC_FLAG  4   [true,true]   (2)
EVALUATION COMPLETE
END  ALogEvaluator: Handling ALog File

BEGIN ALogEvaluator LogicTestSequence =========
LogicTestSequence: Total Aspects:2
Enabled:   true
Evaluated: true
Satisfied: true
Status:    pass
++++++++++ aspect[0] ++++++++++++
Lead Conditions: (1)  WC_FLAG = 3
Pass Conditions: (1)  EFLAG = 0
Fail Conditions: (none)
Enabled(true), Evaluated(true), Satisfied(true), Status(pass)
++++++++++ aspect[1] ++++++++++++
Lead Conditions: (1)  WC_FLAG = 4
Pass Conditions: (1)  EFLAG = 0
Fail Conditions: (none)
Enabled(true), Evaluated(true), Satisfied(true), Status(pass)
END   ALogEvaluator LogicTestSequence =========
Result: pass=true
```