Introduction to using monitoring data for PMT-cuts

-- a brief guide by M. Schimassek (martin.schimassek@ijclab.in2p3.fr), 04.06.2024 --

Google-drive with resources: link

Code-repository: link

Bugs:

- hard-coded paths in python scripts - fixed 04.06.2024
- missing instruction to use python environment with pandas, numpy - added 04.06.2024
- missing instructions to install boost - fixed 04.06.2024

This document is meant as an introduction to using monitoring data to prototype PMT-quality cuts for the LPMTs. We will cover a simple tool for reading the monitoring data from the root-files, the ideas behind some of the proposed cuts, and how to easily analyze the monitoring data.

Note that the tools used here are fundamentally an effort in prototyping proper PMT-quality cuts and alarms which are to be implemented in monitoring and merging respectively. For simplicity of usage, the prototyping doesn't need the tools necessary for e.g. merging and can be done here.

Source Code and Compilation

The source code used for this example can be found in the auger-gitlab. You can clone the repository with

```
git clone --recurse-submodules git@gitlab.iap.kit.edu:auger-observatory/
sandboxes/schimassek-m/uubmonicuts.git
```

if you have an existing gitlab account with ssh-key. If this is not the case, you can find a zip-version of the code (might be out of date) here. It is strongly advised to have a working auger-gitlab account for developing code.

Compilation

A requirement for compiling the C++ code in this repository is an installation of CDAS to read the root-files. You can use the CDAS installation used for offline if you have offline installed.

Should you not have CDAS installed, we refer to the CDAS repository, and ape. You can try installing CDAS using ape with

```
./ape install cdas
```

```
./ape install cdas
```

Should you be unable to install CDAS, you can skip the part of this manual that extracts the monitoring data from the root-files. In the repository, there is already an example file provided called "monit_2023_08_01-sdmc.dat". You can continue with this file.

If you have located your CDAS installation, you can proceed by setting the correct environment for compilation. In the `env.sh`-file you can find the environment-settings for my machines that you can use to create your own.

```
export CDASHOME="/home/schimassek/git/ape/cdas/v6r4p0"
export LD_LIBRARY_PATH="$CDASHOME/lib:$LD_LIBRARY_PATH"
source "/home/schimassek/git/ape/External/root/5.34.00-36f558c7d9/bin/thisroot.sh"
```

You have to

1. You need to define `CDASHOME` as it is used in the Makefile used for compilation. It should be just the directory of your CDAS installation.
2. You have to add the library directory of your CDAS installation to `LD_LIBRARY_PATH` so that linking can work.
3. You have to source the appropriate ROOT used for the CDAS installation. There is usually a script added by the ROOT-installation that you can use.
4. have a boost installation that can be used. For ubuntu systems, you can install it with

```
sudo apt-get install libboost-all-dev
```

With these three steps you should be able to simply type "`make`" to compile the C++ program for reading the mc-root files. You can expect compiler warnings (tested on gcc 11.4) from TMatrixT originating from root-5 if you still are bound to root-5 due to CDAS. You can ignore these warnings.

Running the monitoring data extraction

The next step in preparing the monitoring data we want to analyze in the example, is to use the compiled C++ program "read_monitoring". You can run it simply with

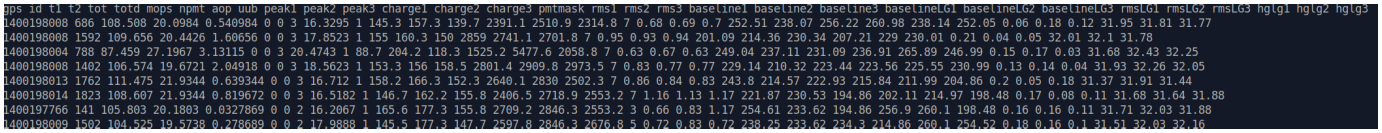
```
./read_monitoring -i /.../mc_2024_01_01.root -o monit_2024_01_01
```

where the "-i" option specifies the input root-files and the -o option gives the basis for the output-file names. One of which is provided for convenience in the google drive folder linked at the beginning of this document. If you are interested in obtaining these files, you can use '`irods`' with

```
iget /pauger/Malargue/Raid/monit/Sd/2024/05/mc_2024_05_01_00h00.root
some_existing_folder/
```

The program should run in about 10 seconds and it will print occasionally "error: no time ... 0" on the standard error-out (std::cerr). You can ignore this warning for now. You should find a file named "monit_2024_01_01-sdmc.dat" after running this program. You can check its content with an editor of your choice (I recommend less to avoid waiting).

You will find a number of ascii numbers:



where the first line is a column description that can be used directly in pandas.

The columns are

- gps: the gps time of the monitoring data send
- id: the local station id that send the data
- t1: the T1-threshold rate in Hz
- t2: the T2-threshold rate in Hz
- tot/totd/mops: the ToT/ToTd/MoPS-rate in Hz
- npmt: the number of non-masked LMPTs
- aop: the average area-over-peak as found from online calibration (currently wrong number)
- uub: boolean flag indicating if it is sent from an uub or not (1 = UUB, 0 = UB)
- peak1/peak2/peak3: the online calibration value of the VEM-peak of LPMT-1/2/3
- charge1/2/3: the online calibrated value of VEM-charge for LPMT-1/2/3; Note: this value is incorrect for now (might be fixed in test-array with new DAQ-versions).
- pmtmask: the actual value of the PMT-mask, which is a bit field: lowest bit → PMT-1, second lowest PMT-2, ...
- rms1/2/3: standard deviation of the baseline (pedestal) for LPMT-1/2/3 as calculated by the FPGA (note: work on-going to understand if this correctly covers the noise)
- baseline1/2/3/: average value of the high-gain baseline of LPMT-1/2/3 (in filtered and downsampled traces (?))
- baselineLG1/2/3: average value of the low-gain baseline of LPMT-1/2/3
- rmsLG1/2/3: standard deviation of the baseline for the low-gain channel of LPMT-1/2/3
- hglg1/2/3: mean value of the high-gain low-gain ratio observed in the LS-DAQ (previously called Anode/Dynode ratio)

It is in principle possible to extract the data of many days at once with this program, simply by specifying all filenames (repeating -i is not necessary). We would recommend to stick to a daily processing as daily values are traditionally used for defining cuts and alarms.

If you want to dive into how these files are extracted, you can have a look into the code. But we do not expect this to be necessary for starting out.

The first look at the monitoring data

For the first look at the data in this example, you can use the ipython notebook "Example_1.ipynb" provided in the repository. To run it, open jupyter-notebooks from the repository and click on the Example_1.ipynb. We use astropy, numpy, matplotlib, and pandas. Make sure that you have these installed or source an environment that has these packages available.

Reproducing the PMT-cuts

Moving away from simple first looks at the monitoring data, we can try to have a look at the proposed monitoring alarm cuts for LPMTs, as described in GAP-2024-025. The main idea is to use mean and RMS of a given value per PMT per day to decide on whether or not a PMT is good.

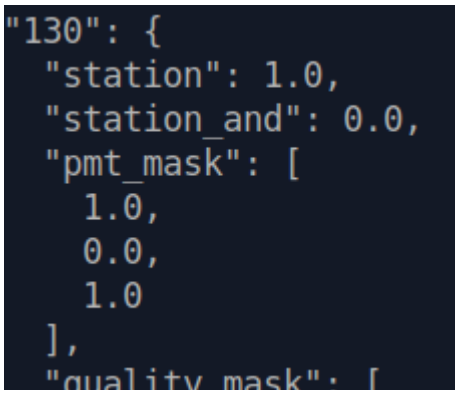
In the repository, there is already the code used for creating the plots of GAP-2024-025 that is also able to create the PMT-mask files that can be used to check the quality selection.

To run it, simply execute (with a correct environment for python: pandas, numpy, matplotlib, see above)

```
./create_moni_pmt_masks.py -i monit_2023_08_01-sdmc.dat -o pmtcuts_2023_08_01
> pmtcuts_2023_08_01.log
```

where piping the stdout into a log-file is not strictly necessary. However, it contains human readable output of what the analysis is doing step-by-step. The output of this analysis should now be a number of pdf-files with plots similar to those in GAP-2024-025 showing distributions of given quantities and the cuts used.

There is also a ".fail" file that contains information of which PMT of each station was rejected and why. The main output information is however the ".masks" file, that compresses the PMT-status information into boolean values. A typical entry in this file (pretty-printed to be human-readable) looks like this:



```
    1.0,
    0.0,
    0.0
  ]
},
```

So for station 130, the first level key in this json-dictionary, we find the station status 1, indicating that at least 1 PMT has passed the quality selection. The "station_and" flag indicates that at least one non-masked PMT failed the selection. The lists "pmt_mask" and "quality_mask" have the information per LPMT. The PMT-mask is taken from the average PMT-mask on the day (rounded to integer) and indicates the status directly from the LS-DAQ. The "quality-mask" tells you which PMT-passed the cuts. If the PMT is masked, also the quality cut is 0, but it is not counted towards the "station_and" status.

If you want to use these station masks for analysis, you can simply read them using the json library:

```
with open(args.masks, 'r') as fin:
    mask_data = json.load(fin)
```

and then for example select only stations with 3-non masked PMTs passing the quality cuts:

```
def extract_mask(mask_data):
    out = {}
    # convert from json data
    for i, dic in mask_data.items():
        if dic["station_and"] > 0:
            out[int(i)] = 1
            m = np.sum(dic["pmt_mask"])
            if m < 3:
                out[int(i)] = 0
        else:
            out[int(i)] = 0
    return out
```

This example is from a simple project to test the Scaler data at https://gitlab.iap.kit.edu/auger-observatory/sandboxes/schimassek-m/uub-scaler-playground/-/tree/main?ref_type=heads

If you want to check visually which stations are passing the quality cuts and which ones don't, you can use the plotting script provided as

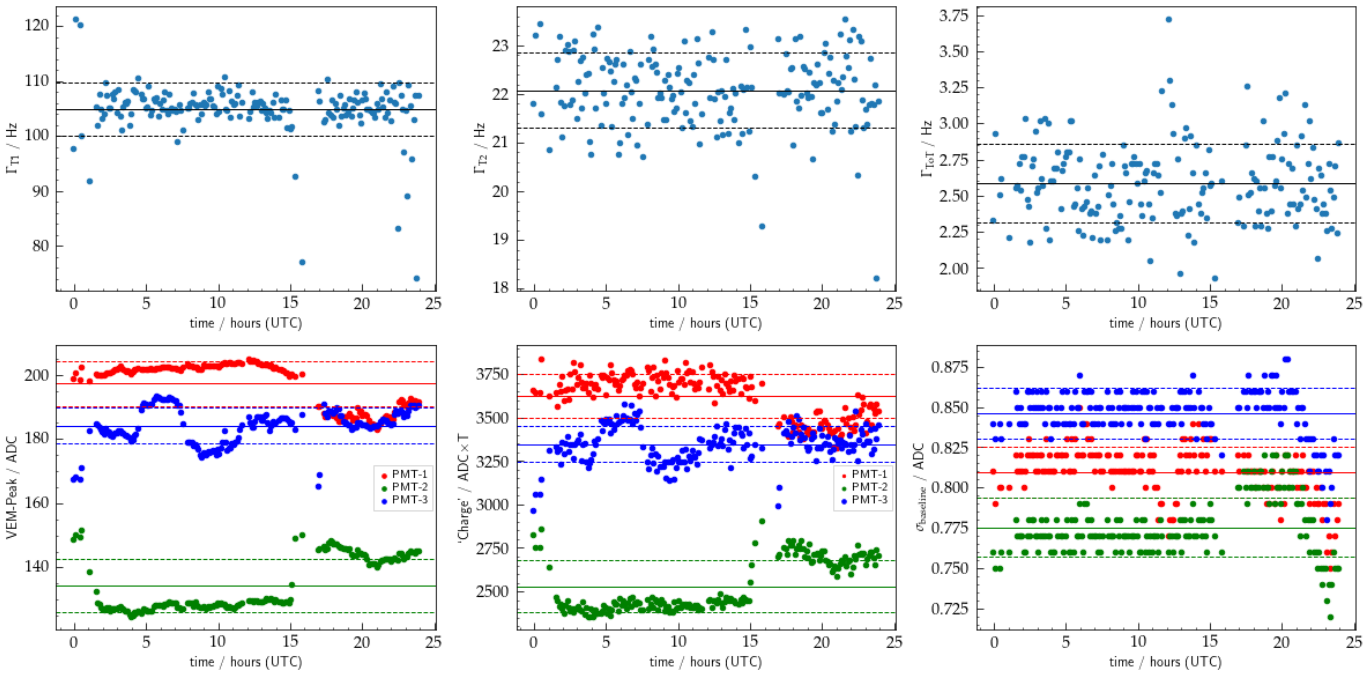
```
./plot_station_masks.py -i pmtcuts_2023_08_01.masks -o plot_2023_08_01
```

the script will also print a summary of accepted / rejected stations.

If you want to check for a given station, how the different quantities are during the day, you can use another script from the repository:

```
./plot_simple_station_checks.py -i monit_2023_08_01-sdmc.dat -o
check_2023_08_01 -s 130
```

where the -s option selects the station in question (here 130):



If you want to check how changing the 'alarm' thresholds will affect the number of stations rejected, you can run the script creating the masks with a different config file:

```
./create_moni_pmt_masks.py -i monit_2023_08_01-sdmc.dat -o pmtcuts_2023_08_01
-c mycuts.json > pmtcuts_2023_08_01.log
```

in the original statement no -c option is provided so that it defaults to the standard one that you can find in the repository as "gap2024_025_cuts.json". The names in the json-file should be fairly clear. If no values are provided for one of the cuts, the script will crash. So you can simply copy the default configuration and adjust the limits to test.