June 19, 2015

Software for the HARPO experiment

Contents

1 Installation

1.1 Requirements

ROOT $\geq$ 5.32.28 CERN data analysis framework. `http://root.cern.ch/drupal/`

libconfig++ On Ubuntu, install the libconfig++-dev package

1.2 Configuration

Several environment variables are required:

`HARPO_USER_DIR` directory where you will install the libraries and binnaries for the HARPO code. It should contain `lib/`, `bin/` and `cfg/` subdirectories.

`HARPO_DATA_DIR` contains raw data.

`HARPO_RECO_DIR` will contain reconstructed data (ROOT files).

`HARPO_ANA_DIR` will contain analysis outputs (ROOT files and pictures).

`$HARPO_USER_DIR/lib` needs to be added to the library path (`$LD_LIBRAY_PATH`. Example of configuration:

```
export HARPO_DATA_DIR=/net/llrdata1.in2p3.fr/data/DATA/data.harpo/2012/Daq
export HARPO_RECO_DIR=/net/llrdata1.in2p3.fr/data/DATA/data.harpo/2012/Reco
export HARPO_USER_DIR=$HOME/Harpo/
export HARPO_ANA_DIR=$HOME/HarpoAnalysis/
export LD_LIBRARY_PATH=${HARPO_USER_DIR}/lib:${LD_LIBRARY_PATH}
export PATH=${HARPO_USER_DIR}/bin:${PATH}
```

1.3 Downloading

The code is available with git:

```
cd YOUR_HARPO_DIRECTORY
git init
git clone https://llrgit.in2p3.fr/harpo/harpo.git
cd Code/Analysis/trunk
```

You can also download a tar ball on `https://llrgit.in2p3.fr/harpo/harpo/tree/master/Code/Analysis`.

1.4 Compiling

The code compiles with make.

```
cd YOUR_HARPO_DIRECTORY/Code/Analysis/trunk
make install
```

2 Usage

For the user, all the relevant files are in the `reco/` directory.

2.1 Making analysis code with the `HarpoAnalysis` class

The HarpoAnalysis class is the base class to process data event by event. It has three main functions:

Init() This is run at the beginning of the analysis. You typically initialize histograms and counters there.

process() This is where the actual data processing is done. `HarpoEvent* fEvt` contains the aggregated data from all the detectors included (Feminos X, Y, PMm2, ...) for a single event (already synchronized).

Save() This is run at the end of the analysis. You can do some final processing there. You will also explicitly save your output histograms in a file defined with the `-b` option (see section 2.2).

A template analysis class `HarpoAnalysisTemplate` exists. The `makeNewAnalysis.sh` script allows to generate a copy of the template with all the required files and modifications to `Makefile` and `harpoanalysis.cxx`.

```
./makeNewAnalysis.sh Name
ls HarpoAnalysisName*
  HarpoAnalysisName.cxx
  HarpoAnalysisName.h
  HarpoAnalysisNameLinkDef.h
```

2.2 Applying the analysis

The `harpoanalysis` program can run your analysis (if you created it with `makeNewAnalysis.sh`). You just need to add the analysis name in a configuration file:

```
# File myConfig.cfg

Analyses="HarpoAnalyseName HarpoAnalyseWriter";

@include "HarpoReconstruction.cfg"
```

The analyses in the list will be applied sequentially for each event.

You can then apply the list of analyses to data:

Raw data (requires the environment `$HARPO_DATA_DIR`)

```
./runRawData harpoanalysis RUNNO -c myConfig.cfg
```

ROOT file `harpoanalysis -c myConfig.cfg --root run.root`

2.3 Running with parameters

Finally, with can run our analysis on some data. The relevant parameters can be added inline, or within a configuration file. The inline commands are as follows:

-h, --help print this text and exit

-v, --verbose increase verbose level can repeat several times

-D, --debug increase debug level can repeat several times

-s NNN , --skip skip NNN events out of NNN+1

-n NNN , --nevents number of events to process

-f NNN , --from read from this event number

-r NNN , --run run number , do not guess from file name

-e NNN , --event number to show

-c FILE, --config config file name

-o FILE, --output output root file name for full data

-b FILE, --hbook output root histograms file name

-d mask, --dets (default 0x3) active detectors mask

-d det, --dets detector id for reader type 'det'

-T type, --type (default 1) 0: read a single detector 1: read real data, 2: generate simulated data, 3: read ROOT file

--det equivalent of -T 0 (single detector)

--raw equivalent of -T 1 (read real data)

--sim equivalent of -T 2 (generate simulated data)

--root equivalent of -T 3 (read root file)

-X type, --xdcc x dcc reader type 0: T2k , 1: MINOS

-Y type, --ydcc y dcc reader type 0: T2k , 1: MINOS

-P type, --pmm2 pmm2 reader type 0: TEXT , 1: WIN, 2: Linux, 3: WIN modified

For example, to process 100 events on data with kit DCC X and Y only:

```
./example -n 100 -d 0x3 -X 0 -Y 0 fileX.dat fileY.dat
```

To run on data with Feminos X and Y and PMm2 (last windows version):

```
./example -d 0x7 -X 1 -Y 1 -P 3 fileX.aqs fileY.aqs filePmm2.dat
```

If you need extra configuration, you can add it in a configuration file `config.cfg`:

```
./example -c config.cfg -d 0x7 -X 1 -Y 1 -P 3 fileX.aqs fileY.aqs filePmm2.dat
```

Where the configuration file can look like this:

```
Pmm2:
{
  nResets = 1; # number of timestamp resets before TPC data
               # All time stamps before the last reset are set to zero
};
Reader:
{
  latency = 1328; # default 1268, 1328 for cosmic+veto mesh
  ppm2Sync  = 3;  # Synchronisation margin between PMM2 and Feminos
};
ana1:
{
  cut1 = 0.;
  cut2 = 500;
};
```

You can access these parameters in you analysis with the following lines:

```
Double_t cut1 = 0;
if ( ! gHConfig->Lookup("ana1.cut1",cut1) )
  Info("InitCfg","Use default fCut1 %g",fCut1);
else
  fCut1 = cut1;

Long64_t cut2 = 0;
if ( ! gHConfig->Lookup("ana1.cut2",cut2) )
  Info("InitCfg","Use default fCut2 %g",fCut2);
else
  fCut2 = cut2;
```

Be careful, the parameters have to be read in 64 bits (Double_t, Long64_t).

3 Monitoring tools

Several monitoring programs were developed to check the quality of the data during tests and data taking. These programs are evolving constantly to adapt to new developments in the system and new observations in the experiment.

3.1 Online monitoring

During data taking we can monitor the data with a semi-online system. The program reads and processes the data that is being produced, but if the data is produced faster than it is processed, the "online" monitor will not be able to show the latest event. This issue can be avoided by using the `-s N` option and process only every `(N+1)`th event.

The online monitoring has three components:

`hrootmonitor` Processes all the data (X, Y and PMm2 synchronised in events, i.e. only triggered events) and produces histograms stored in `/tmp/harpo.map`. This file is always the same, and only one `hrootmonitor` process should run at a time.

`pmm2monitor` Processes all the PMm2 data (regardless of the trigger) and produces histograms stored in `/tmp/harpoPmm2.map`. This file is always the same, and only one `pmm2monitor` process should run at a time.

`hmonitorgui` Reads the histograms in `/tmp/harpo.map` and `/tmp/harpoPmm2.map`, and shows them in a Graphical User Interface (GUI). It runs in principle independently from the HARPO framework. It can be stopped and restarted at any time, without interfereing with the data processing.

If the data is stored with standard naming in $HARPO_DATA_DIR, monitoring (hrootmonitor and pmm2monitor) can be started with the relevant options with the script:

```
monitorRun.sh RUNNO (FIRSTEVT) (S)
```

Where RUNNO is the run number, FIRSTEVT is the number of events to skip at the beginning (optional, default is 0) and S is the number of events to skip (optional, default is 0). This script will also make sure no other instance of the monitoring is already running. If a monitor is already running, and you want to start a different one, you can do it with killMonitor.sh.

Once these processing programs have started, you can start the display (if it is not already running):

```
hmonitorgui
```

Remark: These programs use the TMapFile object in ROOT (http://root.cern.ch/root/html/TMapFile.html). It can encounter memory management issues. If there is a problem (core dump), it might be solved by modifying the prefered memory addresses (see http://root.cern.ch/root/html/TMapFile.html#TMapFile:SetMapAddress). It can be done using the monitor.address and monitor.addressPmm2 fields in the configuration file.

Remark 2: The monitorRun.sh script uses the configuration file defined with the environment variable $HARPO_CFG_FILE. If this variable is not set, it will use $HARPO_USER_DIR/cfg/HarpoReconstruction.cfg, which is created at installation.

3.2 Offline monitoring

The offline monitoring is a quick summary analysis of a run, using reconstruction information (clustering, tracking, matching). The first step is therefore to perform the reconstruction (using the HarpoReconstructionWriter class).

If $HARPO_DATA_DIR, $HARPO_RECO_DIR and $HARPO_ANA_DIR are set correctly, this can be done with the script:

```
monitorOffline.sh RUNNO (FIRSTEVT) (S)
```

Where RUNNO is the run number, NEVTS is the number of events to process (optional, default is 10000), and S is the number of events to skip (optional, default is 0). This will create a ROOT file with the reconstructed data in $HARPO_RECO_DIR/runRUNNO.root, another root file with the monitoring histograms in $HARPO_ANA_DIR/monitor/hist_runRUNNO.root and a summary pdf file in $HARPO_ANA_DIR/monitor/runRUNNO.pdf.

3.3 Electronics noise monitor

We can measure the baseline fluctuations in the electronics. This can only be done when the data is acquired without zero-suppression. If `$HARPO_DATA_DIR` is set correctly, this can be done with the script:

```
monitorNoiseRun.sh RUNNO (NEVTS) (S)
```

Where `RUNNO` is the run number, `NEVTS` is the number of events to process (optional, default is 10000), and `S` is the number of events to skip (optional, default is 0).

4 Data

4.1 Raw data

We collect data from different detectors, with different formats:

DCC `.dat` binary files of the charge map in time (511 time bins) and channels (304 channels, 288 data + 16 noise). Acquired with FEC and DCC kit. The sampling rate, and therefore the length of the time bins is parametrised in the electronics as integer fractions of 100MHz.

Feminos `.aqs` binary files of the charge map in time (510 time bins) and channels (304 channels, 288 data + 16 noise). Acquired with Feminos card. The sampling rate, and therefore the length of the time bins is parametrised in the electronics as integer fractions of 100MHz

PMm2 `.dat` or `.aqs` binary files of the charge collected by each of the 12 PMs, whenever that individual PM was triggered. Acquired with PMm2 card (16 channels, 4 unused).

A run typically produces 3 files: X and Y planes (DCC or Feminos format), and PMm2.

Data from several runs with cosmic rays, taken at LLR with different types of triggers, can be downloaded here (protected with the password "harpo"):

`https://llrbox.in2p3.fr/owncloud/public.php?service=files&t=afa4beff8f10ea7a9aa8eb32276e5015`

4.2 HarpoEvent

The analysis framework collects data for each triggered event into a ROOT TObject of type `HarpoEvent`. It contains the X and Y maps, and the PM data, for one event. The data from the different sources are matched using their timestamps.

The `HarpoEvent` can also store reconstruction information (clustering, tracking, matching) in an object of type `HarpoRecoEvent` (if the reconstruction has been performed. If the event has been produced from simulation, it will also contain Monte Carlo information in a `HarpoSimEvent` object.

These events can be stored in a ROOT `TTree` by using the `HarpoAnalyseWriter` analysis class.

4.3 Data sizes

A complete raw event (X map, Y map and PM data) weighs on average about 15kB with zero suppression, 600kB without. After full reconstruction, including pedestal suppression if needed, and compression in a ROOT file, each event weighs about 50kB.

At a 100Hz event rate, the data production is about 1.5MB/s = 5.4GB/h. The full reconstructed data will be around 20GB per hour of data taking.

5 Simulation

5.1 Short description

A simulation of the HARPO detector is implemented. It is not yet a full description of the detector, and contains the following elements:

event generator This generates "events" that are simply a list of high energy charge particles, with their nature and 4-momentum.

ionisation This describes the ionisation of the gas in the TPC by the charge particle. It can be done with Geant4, which includes the PhotoAbsorption Ionisation model, and multiple scattering in the gas. A simple model is also implemented, describing straight (high momentum tracks) with a poissonian path and a simple ionisation model (power law). This produces a list of ionisation points along the track, with the number of electrons produced.

electron drift Each ionisation electron is treted individually. The electron drift is described simply with a fixed drift velocity, a Gaussian diffusion proportional to the square root of the drift distance, and an exponential law for electron capture.

amplification and readout The signal from each electron is first amplified with gain fluctuations following an appropriate model (for now, only an exponential law is implemented). The signal is then distributed on the readout strips using a spatial distribution function (Gaussian). Finally, the time response of the electronics is describe with a Γ^4 function ($\propto x^4 \exp^{-4x}$).

5.2 Usage

6 Reconstruction

The data processing is done in several seps, in a modular way. First we clean up the data from easily identified noise. Then the signals from clouds of electrons are grouped in clusters. The clusters are used in a tracking algorithm, to extract trajectories on each of the readout map. Finally, the tracks from each map are matched together to create a 3D picture of the event.

6.1 Noise correction

`HarpoAnalyseNoiseSuppression`

6.2 Clustering

`HarpoClustering`

6.3 Tracking

`HarpoHoughTracking`

`HarpoTrackingPh`