ECAL repository - readme and explanations

Eliot Bornand, Kimberly Keyser, Pascal Kolly

October 2023

Contents

1 Structure of code

The repository is composed of multiple folders. Some contain central component of the code, some are specific to experiment (e.g. lifetime computation, time alignment, ...).
The project contains both python script and jupyter notebooks, which are used for specific computations. This project is using object oriented programming.
The different scripts are run in an mamba environment using python 3.12 and need the following package :

```
pandas uprootline_profiler numpy matplotlib mplhep tqdm filterpy uncertainties
```

2 Utils

The folder `utils` contains basic files including : `parameters.py, physics.py` and `data_loading.py`.
The file `parameters.py` declares the geometrical properties of the detector which can be used by all the other sections of the repository.
The file `physics.py` contains some geometrical functions useful for other computations.

2.1 Data loading

The file `data_loading.py` contains the functions required to import the data from the root format.

The `load_run()` function takes as argument the file path of the data runs as well as the minimal and maximal number of hits, acting as a first filter.
This function goes through every root file in the given directory (often of the form "data_00001") and extracts the data using the `load_dataset()` function, storing it in a dataframe. It then concatenates the different files into only one array.

Root format data is stored using tree structure. The `load_dataset()` function import data of the main tree using functions from uproot package. It returns array contain all data and which is labelled with the string of the `br_list_data` (`evt_number, timestamp, n_hits, ....`)

3 Tracking

This section contains the different class used to describe the objects and the functions required to track the muons.

3.1 Hit

The class `Hit` describe a single hit.

This class contains the following attributes :

> `coord` (array of two `float`): contains the 2D coordinate (X,Z) or (Y,Z) depending on if the hit is on a x or y layer of the detector.
>
> `is_sidex (bool)` : boolean which is true if the hit is on a x-layer and false if it is in a y-layer.

`timestamp (float)` : timestamp of the hit (relative time of the hit with respect to the event).

`timestamp_event (float)` : timestamp of the whole event (common to all the hits of this event)

`value (float)` : amplitude value of electronic signal of this hit

The initialisation can take different numbers of arguments (0, 2 or 5). Most of time, these objects are initialised with two arguments. The first argument is a an array containing many hits (who are part of the same event) and the second is the index of the hit we want to consider.

The initialisation with five arguments allows to directly initialisze the 5 attributes in the order presented above.
The initialisation with 0 arguments create an empty instance.

During the initialiation with two arguments, the coordinate are computed from the topfet_id and tofpet_channel with the `mapping_2D()` function located in `track_reconstruction.py`

The class also posses a function `get_pos()` which return the coordinate of the the hit.

It also possess a function `print()` which display the elements of the hit.

3.2 Track

The class `Track` represents a list of hit which composed the same event. The goal of this class is to gather the hits of a same event and to compute the track reconstruction (as mathematical object).

This class contains the following attributes :

`hits` (`Hit`) : list of `Hit`

`t (float)`: tangent angle of the track ($O°$ is a vertical track)

`x0 (float)`: extrapolated coordinate of the crossing of the top of the box

`hits_index (array of int)`: index of the considered hits in the event

`n_freedom (int)` : number of freedom degrees (often size of `Hit` list)

`reduced_chi2 (float)` : χ^2 between the reconstructed track and the list of hits, reduced by the number of freedom degrees,

`mean_time (float)` : unused attribute

It can be initialised of different way (0, 1 or 3 arguments). Most of time, it is initialised with only one argument, which is the list of `Hit`. This list of Hit become then the attribute `hits` of the Track instance.

The initialisation with three arguments allows to directly initialise this two geometrical attributes `t` and `x0` in addition to the list of hits.
In case of an initialisation with 0 arguments, an empty instance is created.

During initialisation with 1 argument, the function `find_track()` is used to generate a track from the list of hits. This function create a track using the Houd transformation or an other method. The function can take as optional argument a bool to plot the track. This function compute and initialise the attributes `t` and `x0`. It then call the `get_track()`, `get_indices()` and `chi2()` functions.

> The function `get_track()` compute the coordinate (x,z) or (y,z) of the track (as a mathematical function).
>
> The function `get_indices()` compare the reconstructed track and the list of hit and initialize the attribute hits_index with the indices of the hits belonging to the track (coordinates of the track are rounded to match with the coordinates of hits). It takes as argument the reconstructed track, if nothing is given, it computed it from the the list of hits of the instance.
>
> The function `chi2()` compute the χ^2 between the track and the list of hit. The function `is_good_fit()` check if $\chi^2 < 3.841$, which can be used as a criteria of the quality of the track reconstruction.

The function `keep_hits_only()` sort the list of hits and keep only the one corresponding with `hits_index`.

The class is equipped of a series of function returning information's of the hits such as :
- `get_hits_pos()` (return the `coord` of each element of `hits`)
- `get_hits_coord()` (same as previous)
- `x()` (for a given z coordinate, returns the corresponding x coordinate on the track)
- `get_time_interval()` (returns the mean timestamps of the hits)
- `_dr()` (returns the spatial distance between a reference hit and a list of hits)
- `get_timestamp()` (returns `timestamp` + `timestamp_event` for each element of `hits`)

The class has a `kalman_filter()` function which returns the positions of hits after applying a Kalman filter (it uses the `KalmanFilter` class of the `filterpy` package).

There is also a `print()` which print the main attributes of the instances and can display the track by calling an other function, the `get_plot()` function.

3.3 Track 3D

The class `Track3D` represents a track in 3 dimensiosn (x,y,z). Its structure is two `Track` instances, one with on (x,z) plane and the other on (y,z) plane.
This class contains the followin attributes :

`x (Track)` : List of hits from x-layers.

`y (Track)` :List of hits from y-layers

`time` (array of `float`) : List of timestamps of each `Hit` of the track 3D.

It can be initialised in different way (0,1,2). The initialisation with 1 argument takes a list of `Hit` and sort them with respect to `is_sidex` in two lists. It then create initialise the attribute `x` and `y` with the two lists of hits.

The initialisation with 2 arguments allows to directly initialise the two `Track x` and `y` with the arguments.
The initialisation with 0 argument create an empty instance.

The class is equiped with function `get_time_interval()`, `kalman_filter()`, `find_track()`, `reduced_chi2()` adapted to a track 3D. The function `is_good_2D_fit()` call the `is_good_fit()` function of `Track` for both x and y tracks.

The class has `print()` function which display individuals plots of both x and y tracks. It also has a `show()` function which display a three dimensional plot of the entire track 3D.

3.4 Track reconstruction

The script `Track_reconstruction` contains some useful functions to manipulate class `Hit`, `Track` and `Track3D`. Some functions are kind of duplicates of them contained in the class definitions.

The function `mapping_2D()` makes the conversion between the mapping in term of tofpet_id and tofpet_channle to the spatial mapping in term of layers and bars. It uses`is_sidex()` which indicate it a tofpet_id is on side x or y. The function `mapping_inv_2D()` makes the inverse conversion.

The function `mapping_SiPM_delay()` map the delay (time for the electronic signal to travel through the board, in picosecond) of each SiPM channel to the tofpet id and channels. It uses `SiPM_delay` in `parameters`, and then convert the delay (in picosecond) into clockcycle using the function `convert_ns_to_clockcycle()`.

The `max_overlap()` function looks how many hits overlap at a certain angle t. It returns the the hits index that overlap, the number of overlaping and the boundaries and the boundaries (are the extremal x that belongs to the overlap region).

4 Time alignment

The time alignment folder contains what is necessary for the time alignment procedure. It has for purpose to compute the general offset of each channel, to compute the time resolution of the experiment and to provide a set of functions for applying time correction to all future data

4.1 Time correction

The `time_correction_fiber()` function corrects the time that the light take to travel until the photon detector inside the fiber. It computes the distance travelled by light from the spatial position of the hit and the speed of light in the fiber (15 cm/ns). It can be called with a Track3D in argument, as the spatial position of the hit is required and can only be fully determined from a reconstructed track. If it is called with a Track3D and a Hit, it compute the time correction for the Hit's timestamp computing is spacial location from the track reconstruction of th Track3D. This approximation is usefull to compute time correction of electronic shower (which don't have structure of track), considering the shower close to the end of track (or in his continuity).

The `time_correction_electronics()` function correct the time that the electronic signal takes to travel into the board from the SiPM to the central tofpet. It uses the function `mapping_SiPM_delay()` to map the SiPM channel and the corresponding delay time from the tofpet id and channel. If it is called with 3 argument (timestamp, tofpet id, tofpet channel). it changes the modify the given timestamp (relatively to his location) and returns it. If it is called with 1 argument(Track3D), it changes the timestamps of each hits of the track and returns the track.

The `time_correction_offset()` function suppress the general offset of a time measurement depending on the tofpet mapping. It uses a set of offset values computed previously with a time alignment procedure. If it is called with 3 argument (timestamp, tofpet id, tofpet channel). it changes the modify the given timestamp (relatively to his location) and returns it. If it is called with 1 argument(Track3D), it changes the timestamps of each hits of the track and returns the track.

The `time_correction_global()` function applies successively the three previous time correction functions. If it is called with on argument whcih is a Track3D, it applies all corrections on the argument. If it is called with 2 argument, a Track3D and a list of hits, it applies all time corrections on all hits of the list. This is used to compute time correction on electronic shower (which is a list of hits), and required the track that preceded it to compute the fiber correction.

4.2 Find muon track

The file `find_muon_track.py` contains the function `find_muon_track()`. This function is used to select the appropriate muon tracks needed for the time alignment of the detector. It chooses "good" tracks that traverse the whole detector and verifies that the first two and last two layers are each hit only once. This function takes as argument two dataframe of events (the one first sorted by the data loading and the total dataframe with all events). This function use different selection criterion to sort the good track. Only event with 8 hits on each side (so 16 in total) are kept. It also verify to have one and only hit on the first and last bars. The function create then different files in a similar way as `find_muon_decay()` (c.f. Section 5.1).

4.3 Data creation time

The jupyter notebook `data_creation_time.ipynb` calls the the function `find_muon_decay()` to create the required files. The cells in the second part of the notebook are made to plot some tracks.

4.4 Time distribution

This Jupyter notebook `time_distribution.ipynb` load previously filtered data using `data_creation_tim` and create the tracks corresponding to the data. More precisely, it creates three arrays of track containing each steps of the correction : `uncorrected_tracks`, `fiber_corrected_tracks`, `ttracks`. Note that `ttracks` are the final electronics and fiber corrected tracks, it is called with two "t's" in order to avoid some shadowing issue.

Once the data is loaded, these vectors are saved in `pickles` in order to avoid future unnecessary computations and then the notebook proceeds to compute the gaussian offsets and spread for each channels. These values, which are stored in the matrices `muXF`, `muYF`, `sigmaX` and `sigmaY` are then also saved in `pickles`.

Several plotting cells are also present in the folder in order to observe the computations and data.

5 Muon decay

The muon decay folder contains what is necessary to compute the mean lifetime of muons. Additionnaly to the scrpit and notebook usefull to compute the mean lifetime, the subfolder `extracted_data` contains all the file produced by the process.

5.1 Find muon decay

The file `find_muon_decay.py` contains the function `find_muon_decay()`. This function is used to select the appropriate tracks to compute the muon lifetime.

This function takes as argument two data frame of events (the one first sorted by the data loading and the total data frame with all events). It also take as arguments different parameters, usefull to refine the criterion (`time_min, time_max, spacial_min, spacial_max, spacial_cutoff`), to indicate what need to be saved (`save_inideces, save_time_intervals, save_hits, save_stats, save_distances, return_stats`), path and directory information (`run_name, storage_dir`) and finally the boolean `time_corr` indicating if a time correction need to be applied on the datas. The role of the differents arguments are described in the function.
It selects track with a minimum of 3 hits in the x-layer and 3 hits in the y-layer. It then verifies no hits were detected on the side of the detectors or the lower layer. This ensures the track ends inside the detector. Another criteria is for the track to be "good", which means has a good χ^2 value. Finally, the code checks if the next event is located in a close radius of the end of the muon track and below it. Only the time intervals inferiors to a limit are kept, and will be used to compute ht mean lifetime afterward.

The function can produces different files. The `time_intervals.txt` with all the time intervals between muons trcaks and electronci shower. The `event_index.txt` which contains all the `candidate_index` of the selected events (wrt to the total data frame) The `distancesX/Y.txt` which contains the spacial minumal distances between the end of muosn tracks and the closest hit of the next event (for both x and y side). For each criterion applied, statistics of the number of rejected event are computed an can be returned by the function. This different markers are described in the functions. The file `filtering_data.pkl` contains all of these statistics on the selection. Finally the file `pickle_decay_data.pkl` contains all the selected datas from the selection process.

5.2 Raw data analysis

The Jupyter notebook `Raw_data_analysis.ipnyb` make the initial analysis on all datas the select decay events. It first extract root datas from the desired location. It then calls the `find_muon_decay()` function to create the required files (`time_intervales.txt, distances-txt`, ...) for the computation of mean lifetime. The seconde part of the notebook use the indices of the good candidates from the file `events_indices.txt` to plot figures of the selected events which sould be muon decay.

5.3 Compute Lifetime

The Jupiter Notebok `compute_lifetime.ipynb` contains the code to compute the muon lifetime and the plot associated. It needs the path to the `"time_interval.txt"` files in the extracted data folder. It extract the datas and can then apply another selection of minimal/maximal time interval or minimal/maximal radius (between end of muon track and closest hit of electronic shower). It initialise a exponential function usefull afterward to fit eht datas. It then plots the decay time distribution and computes the lifetime following the exponential fit.
The second part of the notebook allows to analyse the variations of the paramaters (`t_min, t_max, r_min, r_max`) and plot figures in order to observe the influence of this paramters on the muon mean lifetime.
Int the third part of the notebook, the zfit package also plots the semi log histogram of the time intervals between the muon arrival and its decay, as well with an exponential fit. This second analysis of mean lifetime is a bit preciser but take more time.

5.4 Analyse decay data

The Jupiter Notebook `analyse_decay_data_ipynb` plots the histogram of the electron hits value. It needs the path to the extracted data folder. pcb xxx

6 Other folders

The repository contains other folders which have been lees used in the context of time aligneemnt and muon's mean lifetime computation. The folder `geatn4` contains the what required to run the Geant4 simulation (c.f D.Berezovska, L. Hartman, N. Glardon, *Cosmic ray detection with an electromagnetic calorimeter*, 2023, for more details). The subdolder `theory` contains class to modelise the displacement of particlue into the detector in order to integrate the Bethe-Bloch function. The subfolders `tests` and `test_import_root` contains various scripts used to test algorythms and functions.