2-low-level-processing

June 24, 2020

1 LOW LEVEL ANALYSIS

1.1 Peak Finding

Come prima cosa viene consigliato di guardare questa nota sulla ricerca dei picchi Peakfinding in strax che spiega come funziona in strax la ricostruzione dei segnali dei PMT e l'identificazione in S1 e S2. Qui c'è un riassunto della nota.

Gli step che vengono seguiti sono:

1. Cercare i **peaklets**: regioni di tempo che appartengono allo stesso S1 o S2
2. Calcolare le proprietà di base e classificarli come S1 o S2
3. Unire peaklets parte dello stesso S2

1.2 1-HitFinder

Prima della ricerca dei picchi viene runnato l'**hitfinder**, che deve cercare single-PE e altri **hits**, gli algoritmi utilizzati cercano di ottimizzare l'accettanza dei fotoni.

1.3 2-PeakLets

Poi gli hits vengono raggruppati nei **peaklets** quando ci sono dei gap >= 350 ns. In questo modo i peaklet possono essere:

- S1 separati da possibili After Pulses (AP) o altri SE che seguono
- S2 senza gap
- Single electrons (un-fragmented)
- Hits isolati dovuti a dark counts o PMT afterpulses
- gruppi non risolti a high-energy

1.4 3-GoodnessOfSplit

Questo non è sufficiente, soprattutto ad alte energie dove possiamo avere APs o PEs. Per i picchi candidati viene calcolata la waveform di somma e viene valutata il **goodness of split**, poi viene separata al massimo se questo supera un certo threshold, quesìoperazione viene ripetuta fino a che on si raggiunge un'area piccola o non si ranggiunge un limite. Strax ha 2 opzioni per il goodness of split, ma viene usata quella chiamata **natural breaks** che guarda la somma quadratica della deviazione del picco(i) prima e dopo lo split. Il goodness of split threshold può essere funzione dell'area e del risetime dei picchi, in questo modo può anche classificare S1 e S2 (che però viene fatto dopo). Qui è descritto il threshold usato attualmente.

1.5 4-Classification

La classificazione è basata su: * **Rise time**: tempo per arrivare al 10% dell'area del picco * **Tight coincidence**: numero di PMT che contribuiscono entro una finestra di 100 ns centrata al massimo della waveform somma

I picchi sono classificati come unknown, S1 o S2 seguendo questo ragionamento: * picchi con risetime breve (<60ns (<150ns) se l'area è meno (più) di 100 PE) e almeno 3 PMTs sono classificati come S1 * i picchi rimanenti con almeno 4 PMT sono classificato come S2 * altrimenti come unknown

1.6 5-Merging

Il merging serve serve per convertire peaklets in **peaks** ed è applicato nei seguenti casi: * il merged peak non ha gap tra hits maggiori di 3.5 us * il primo peaklet è un S2 * l'area del merged peak è minore di 5000 PE * il tempo del merged peak è minore di 10 us

1.7 Peakfinding: peaklet_processing.py

Ci sono 4 classi nel codice:

- Peaklets
- PeakletClassification
- MergedS2s
- Peaks

[1]:
```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from scipy import stats
import straxen
from multihist import Histdd, Hist1d

# Print out exactly what versions we are using, for reference / troubleshooting:
import sys
import os.path as osp
```

[2]:
```
st = straxen.contexts.xenon1t_dali()
run_id = '180215_1029'
```

1.8 Peaklets

Cerca gli hits e li ordina nel tempo, poi usa il gap threshold per formare dei gruppi di hits.

Poi viene calcolata la waveform somma e la larghezza del picco.

Viene fatto lo split usando i natural breaks.

[3]:
```
st.data_info('peaklets')
```

[3]:
```
                   Field name        Data type  \
0                        time            int64
```

```
1                      length            int32
2                          dt            int16
3                     channel            int16
4                        type             int8
5                        area          float32
6             area_per_channel  ('<f4', (248,))
7                      n_hits            int32
8                        data  ('<f4', (200,))
9                       width   ('<f4', (11,))
10  area_decile_from_midpoint   ('<f4', (11,))
11          saturated_channel  ('<i2', (248,))
12       n_saturated_channels            int16
13          tight_coincidence            int16
14                    max_gap            int32
15     max_goodness_of_split          float32

                                          Comment
0                    Start time since unix epoch [ns]
1                   Length of the interval in samples
2                            Width of one sample [ns]
3                                  Channel/PMT number
4                     Classification of the peak(let)
5                       Integral across channels [PE]
6                           Integral per channel [PE]
7   Number of hits from which peak was constructed…
8             Waveform data in PE/sample (not PE/ns!)
9   Peak widths in range of central area fraction …
10  Peak widths: time between nth and 5th area dec…
11             Does the channel reach ADC saturation?
12                 Total number of saturated channels
13                    Hits within tight range of mean
14          Largest gap between hits inside peak [ns]
15                 Maximum interior goodness of split
```

[4]:
```
peaklets = st.get_array(run_id,['peaklets'])
print(peaklets['area'])
```

```
[2.6191945 2.4241385 3.2993686 … 8.369365  2.3876238 1.8463116]
```

1.9 PeakletClassification

Classifica come unknown, S1, or S2. Dipende da peaklets

Vengono calcolati rise_time e numeri di PMT come: * -peaks[‘area_decile_from_midpoint’] * n_channels = (peaks[‘area_per_channel’] > 0).sum(axis=1)

se il risetime è minore di ‘s1_max_rise_time’ (o ‘s1_max_rise_time_post100’) e il numeri di canali è almeno ‘s1_min_coincidence’ allora viene classificato come S1

Altrimenti se il numero è almeno 's2_min_pmts' allora è S2.

[5]:
```
st.data_info('peaklet_classification')
```

[5]:
```
  Field name Data type                         Comment
0       time     int64  Start time since unix epoch [ns]
1     length     int32 Length of the interval in samples
2         dt     int16           Width of one sample [ns]
3    channel     int16                 Channel/PMT number
4       type      int8   Classification of the peak(let)
```

Non viene distinto dove si trovano i PMT che hanno gli hits.

Non c'è una condizione per classificare in base all'area dei picchi.

1.10 MergedS2s

Serve a fare il merge dei peaklets. Dipende da peaklets e peaklet_classification.

Se la variabile 's2_merge_max_gap' è minore di zero allora non viene fatto il merge (dipende dal gap).

Trova i gruppi di peaklets separati e va il merge usando le impostazioni in 'get_merge_instructions'.

1.11 Peaks

Dipende da peaklets, peaklet_classification e merged_s2s

Rimuove i fake merged S2 e ordina i picchi nel tempo.

1.12 Processamento: pulse_processing.py

Nel codice c'è un'unica classe PulseProcessing. Come prima cosa divide i raw_records in: * tpc_records * aqmon_records

Per i TPC records applica il processing di base: 1. inverte, sottrae la baseline e integra la waveform (strax.zero_out_of_bounds, strax.baseline, strax.integrate)

2. applica il software HE veto (software_he_veto)
3. Trova gli hits prima di filtrare (strax.find_hits)
4. applica un filtro lineare per concentrare i PMT pulses: strax.filter_records(r, np.array(self.config['pmt_pulse_filter']))
5. taglia fuori dagli hits (strax.cut_outside_hits)

[6]:
```
st.show_config('records')
```

[6]:
```
                    option                   default  \
0            hev_gain_model          (disabled, None)
1          baseline_samples                        40
2       tail_veto_threshold                         0
```

```
3          tail_veto_duration                    3000000
4        tail_veto_resolution                       1000
5     tail_veto_pass_fraction                       0.05
6       tail_veto_pass_extend                          3
7            pmt_pulse_filter                       None
8           save_outside_hits                    (3, 20)
9                 n_tpc_pmts                   <OMITTED>
10  check_raw_record_overlaps                       True
11     allow_sloppy_chunking                      False
12         hit_min_amplitude  pmt_commissioning_initial
13               pax_raw_dir            /data/xenon/raw
14           stop_after_zips                          0
15          events_per_chunk                         50
16        samples_per_record                        110

                                              current  \
0   (to_pe_per_run, https://raw.githubusercontent…
1                                           <OMITTED>
2                                              100000
3                                           <OMITTED>
4                                           <OMITTED>
5                                           <OMITTED>
6                                           <OMITTED>
7   (0.012, -0.119, 2.435, -1.271, 0.357, -0.174, …
8                                              (3, 3)
9                                                248
10                                             False
11                                              True
12                                         XENON1T_SR1
13                                          <OMITTED>
14                                          <OMITTED>
15                                          <OMITTED>
16                                          <OMITTED>

                              applies_to  \
0   (records, veto_regions, pulse_counts)
1   (records, veto_regions, pulse_counts)
2   (records, veto_regions, pulse_counts)
3   (records, veto_regions, pulse_counts)
4   (records, veto_regions, pulse_counts)
5   (records, veto_regions, pulse_counts)
6   (records, veto_regions, pulse_counts)
7   (records, veto_regions, pulse_counts)
8   (records, veto_regions, pulse_counts)
9   (records, veto_regions, pulse_counts)
10  (records, veto_regions, pulse_counts)
11  (records, veto_regions, pulse_counts)
```

```
12  (records, veto_regions, pulse_counts)
13                        (raw_records,)
14                        (raw_records,)
15                        (raw_records,)
16                        (raw_records,)

                                                 help
0   PMT gain model used in the software high-energ…
1   Number of samples to use at the start of the p…
2   Minimum peakarea in PE to trigger tail veto.Se…
3               Time in ns to veto after large peaks
4   Time resolution in ns for pass-veto waveform s…
5   Pass veto if maximum amplitude above max * fra…
6   Extend pass veto by this many samples (tail_ve…
7   Linear filter to apply to pulses, will be norm…
8   Save (left, right) samples besides hits; cut t…
9                                 Number of TPC PMTs
10  Crash if any of the pulses in raw_records over…
11  Use a default baseline for incorrectly chunked…
12  Minimum hit amplitude in ADC counts above base…
13                   Directory with raw pax datasets
14        Convert only this many zip files. 0 = all.
15               Number of events to yield per chunk
16                      Number of samples per record
```

[7]:
```
st.data_info('records')
```

[7]:
```
             Field name        Data type  \
0                  time            int64
1                length            int32
2                    dt            int16
3               channel            int16
4         pulse_length            int32
5             record_i            int16
6                 area            int32
7      reduction_level            uint8
8              baseline          float32
9         baseline_rms          float32
10  amplitude_bit_shift            int16
11                 data  ('<i2', (110,))

                                        Comment
0                   Start time since unix epoch [ns]
1                  Length of the interval in samples
2                          Width of one sample [ns]
3                                Channel/PMT number
4   Length of pulse to which the record belongs (w…
```

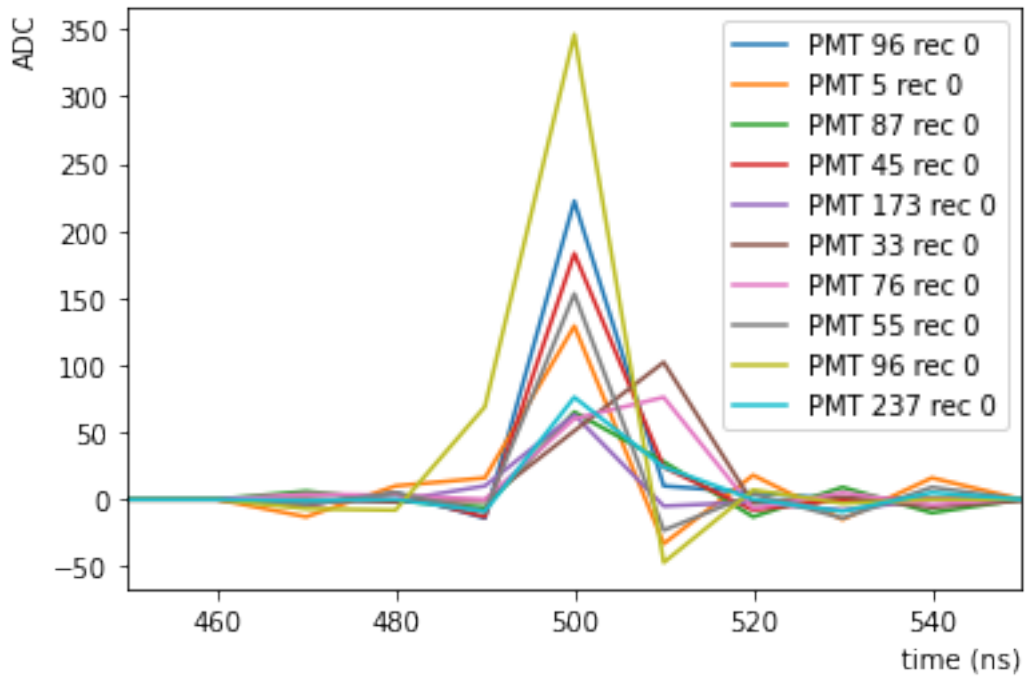
```
5                          Fragment number in the pulse
6                       Integral in ADC counts x samples
7   Level of data reduction applied (strax.Reducti…
8   Baseline in ADC counts. data = int(baseline) -…
9   Baseline RMS in ADC counts. data = baseline - …
10  Multiply data by 2**(this number). Baseline is…
11  Waveform data in raw counts above integer part…
```

[9]:
```
records = st.get_array(run_id,['records'])
```

[43]:
```
dt = records['dt'][0]
print(records['data'].shape[1])
dts = np.arange(0,records['data'].shape[1]*dt,dt)
for i in range(10):
    plt.plot(dts,records['data'][i],label=f"PMT {records['channel'][i]} rec␣
 ↪{records['record_i'][i]}")
plt.legend()
plt.xlabel("time (ns)", ha='right', x=1)
plt.ylabel(f"ADC", ha='right', y=1)
plt.xlim(450,550)
```

```
110
```

[43]:
```
(450.0, 550.0)
```



2 TUTORIAL

Definizioni varie: * channel * self-trigger * pulse * record: * hit: tempo nel quale il record supera il threshould * peak: S1 o S2 candidati * peaklet: candidati di peak * ADC count * PE (photo-electron): in XENON 1 PE è l'area media prodotta da un fotone (long-wavelength) che produce un segnale nel PMT

Nello scorso tutorial abbiamo visto tutto quello che è sopra peaks, questa volta si guarda quello che c'è sotto.

Il **DAQ reader** determinare quali chunk salvare e li divide in dati della TPC, del Muon Veto o Neutron Veto.

Il **Pulse Processing**, calcola la baseline, cerca gli hits e taglia al di fuori di questi, fa il filtraggio e applica il software HEV?

Dopo con il **Peaklet Processing** si cercano e classificano i peaklet per creare i peaks.

2.1 Esempio: processamento di run

raw_records: 83 GB raw, 25 GB on disk Prende più tempo per fare il pulse processing (50%), in particolare il **filtering** (a causa delle convoluzioni). Le operazioni con i peaklets occupano circa il 17% del tempo, mentre le analisi di alto livello sono molto più veloci.

In strax i records sono salvati con la stessa lunghezza, se necessario vengono frammentati, questo per evitare oggetti di diversa lunghezza.

3 Presentazioni one-slide su una funzione del processing

3.1 chiamate in pulse_processing.py e definite in strax/processing/pulse_processing.py

- **baseline** (Sophia A.): prende il record, calcola il valore della baseline facendo la media dei primi 40 campioni, non fa la sottrazione soltanto se non è il primo frammento; poi fa la sottrazione e inverte
- **find_hits** (Daniel): cerca gli hit, fa un loop su tutti i record, quando supera il threshould (ce ne sono 2 tipi, uno del DAQ e uno calcolat dall'RMS della baseline)
- **cut_outside_hits** (Francesco T.): è chiamato appena dopo find_hits, vengono mantenuti 20 ns prima degli hit e 150 ns dopo
- **integrate_lone_hits** (Darryl): hits che non sono nei peaklets, cerca il picco più vicino sia prima che dopo; l'obiettivo di questo algoritmo è essere sicuri di aver calcolato l'area corretta per i picchi

3.2 chiamate in peaklet_processing.py:

- **sum_waveform** (Daniel): fa la somma di più peaklets; come prima cosa produce un buffer lungo il doppio del peaklet più lungo, poi fa il loop su tutti i peaklet; c'è l'opzione store_downsampled_waveform che viene usata se il buffer è inferiore alla lunghezza del record;

[ ]: