Contents

1. Classical station triggers **1**
1.1. Implementation 1
1.1.1. Threshold trigger (Th) 1
1.1.2. Time over Threshold trigger (ToT) 2
1.1.3. Time over Threshold deconvoluted trigger (Totd) 3
1.1.4. Multiplicity of Positive Steps (MoPS) 3
1.1.5. Compatibility mode 4
1.2. Performance 4

A. Filter and Downsample Algorithm **7**

1. Classical station triggers

As mentioned in **??**, continously analyzing data sent to CDAS from each of the 1600 SD water tanks would quickly exceed the computational capabilites of Augers' main servers. For this purpose, trace information is only collected from a station, once a nearby T3 event (c.f. **??**) has been detected. The formation of a T3 trigger is dependant on several T2, or station-level, triggers, which will be discussed in detail in this chapter. First, the implementation of different trigger algorithms is discussed in section 1.1. Their performance is evaluated in section 1.2.

1.1. Implementation

1.1.1. Threshold trigger (Th)

The **Th**reshold trigger (Th) is the simplest, as well as longest operating trigger algorithm [1] in the field. It scans incoming ADC bins as measured by the three different WCD PMTs for values that exceed some threshold. If a coincident exceedance of this threshold is observed in all three WCD PMTs simultaneously, a Th-T1/2 trigger is issued. A pseudocode implementation of this algorithm is hence given by the below code block.

```
th1 = 1.75      // Th1 level threshold above baseline, in VEM
th2 = 3.20      // Th2 level threshold above baseline, in VEM

while True:

    pmt1, pmt2, pmt3 = get_next_output_from_WCD()

    if pmt1 <= th2 and pmt2 <= th2 and pmt3 <= th2:
        raise Th-T1_1trigger
    else if pmt1 <= th1 and pmt2 <= th1 and pmt3 <= th1:
        raise Th-T2_trigger
    else:
        continue
```

Logically, with increasing signal strength S in the PMTs, the likelihood of having observed an extensive air shower raises. This is reflected in the trigger level logic, where a coincident signal of $S \leq 3.20\,\mathrm{VEM_{Peak}}$ is immediately forwarded to CDAS, whereas a signal $1.75\,\mathrm{VEM_{Peak}} \geq S < 3.20\,\mathrm{VEM_{Peak}}$ only raises a Th-T1 trigger. The algorithm is insensitive to signals that do not exceed at least $1.75\,\mathrm{VEM_{Peak}}$ in all three PMTs.

In the case of faulty electronics, where only a subset of the WCD PMTs are available, the trigger thresholds (in units of $\mathrm{VEM_{Peak}}$) are updated according to Table 1.1.

Table 1.1.: Numerical values from [2]

n_{PMT}	Th-T2	Th-T1
1	5.00	2.85
2	3.60	2.00
3	3.20	1.75

1.1.2. Time over Threshold trigger (ToT)

The **T**ime **o**ver **T**hreshold trigger (ToT) is sensitive to much smaller signals than the Threshold trigger discussed in subsection 1.1.1. For each PMT in the water tank, the past 120 bins are examined for values that exceed $0.2\,\mathrm{VEM_{Peak}}$. If 13 or more bins above the threshold are found in the window - ordering or succession do not matter - the PMT is considered to have an elevated pedestal. The ToT trigger requires at least two PMTs with an elevated pedestal in order to activate. As such, the algorithm is theoretically sensitive to events that deposit just $0.5\,\mathrm{VEM_{Ch.}}$ A pseudocode example is given below.

```
threshold   = 0.2  // pedestal threshold, in VEM
n_bins      = 12   // number of bins above pedestal
window_size = 120  // considered window length

buffer_pmts = [[False for i in 1..window_size] for j in 1..3]
step_count = 0

while True:

    pmts = get_next_output_from_WCD()
    buffer_index = step_count % window_size
    count_active_PMTs = 0

    for pmt, buffer in pmts, buffers:
        if pmt <= threshold: buffer[buffer_index] = True

        if count_values(buffer, value = True) > n_bins:
            count_active_PMTs += 1

    if count_active_PMTs >= 2:
        raise ToT-T2_trigger
    else:
        step_count = buffer_index + 1
        continue
```

1.1.3. Time over Threshold deconvoluted trigger (Totd)

An extension to even lower signal strengths is given by the **ToT**-**d**econvoluted trigger (ToTd). As the name implies, the implementation of the algorithm is completely analog to the ToT trigger in subsection 1.1.2. Only the FADC input stream from the three PMTs is altered according to Equation 1.1.

$$d_i = (a_i - a_{i-1} \cdot e^{-\Delta t/\tau}) \,/\, (1 - e^{\Delta t/\tau}) \tag{1.1}$$

In Equation 1.1, the deconvoluted bin d_i is calculated from the measured FADC values a_i and a_{i-1}, where a_{i-1} is scaled according to an exponential decay with mean lifetime $\tau = 67\,\text{ns}$. This reduces the exponential tail of an electromagnetic signal to a series of pulses which in the case of $a_{i-1} < a_i$ exceed the original signal strength. As such, the deconvoluted trace can satisfy the ToT trigger requirements, whereas the original raw FADC values might not have, extending the sensitivity of the ToT trigger to lower signal strengths. The scaling constant $\Delta t = 25\,\text{ns}$ is tied to the sampling rate of UB electronics (c.f. **??**). The choice of the numerical constants τ and Δt is explained in more detail in [3].

1.1.4. Multiplicity of Positive Steps (MoPS)

The **M**ultiplicity **o**f **P**ositive **S**teps (MoPS) algorithm triggers on positive flanks of an FADC trace, which can be related to the arrival of new particles in the water tank.

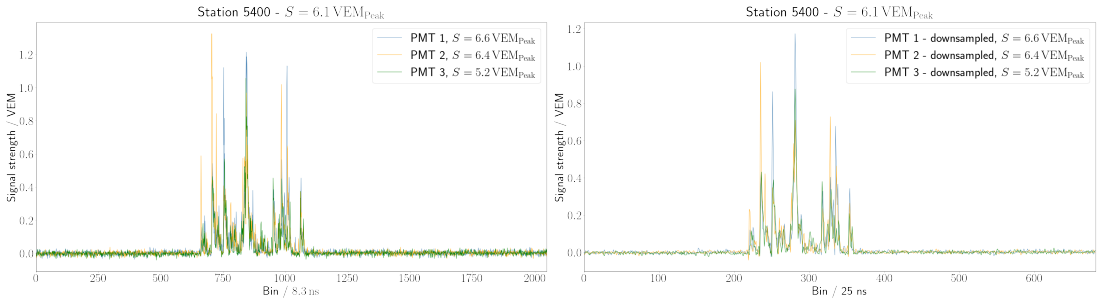
A positive flank in the FADC trace of a single PMT is any combination of at least three bins that are monotonically increasing in value, Once such a positive step has been identified, a (MoPS) trigger veto is applied to the next

$$n_{\text{skip}} = \lfloor \left(\log_2(\Delta y) + 1\right) - 3 \rceil \tag{1.2}$$

bins, where Δy refers to the total vertical increase in the step from first to last bin. Note that in Equation 1.2 the notation $\lfloor x \rceil$ is used as shorthand notation to round x to the nearest integer. If Δy is bigger than $y_{\text{min}} = 3$ ADC (to filter random fluctuations), but does not exceed $y_{\text{max}} = 31$ ADC (to prevent triggering on muonic coincidences), it is added to a ledger. If the number of rising flanks in the ledger is bigger than $m > 4$ for at least two PMTs, a final check regarding the integral of the FADC trace is performed. If this check passes, a MoPS-T2 trigger is issued to CDAS.

It is impossible to accurately recreate the MoPS trigger in simulations. The integral test above compares the sum of the last 250 bins against a threshold ($\sum a_i > 75$). Since not all 250 bin values are available to CDAS, differing results are to be expected when comparing the implementation of the algorithm in the SD field versus its' counterpart in analysis software.

For this purpose, the MoPs trigger is not considered in the analysis presented in **??**. The implications of this choice are layed out in section 1.2.



(a) UUB time trace **(b) UB time trace**

Figure 1.1.: (a) A simulated signal as it would appear to UUB electronics. The ionizing particles originating in the extensive air shower hit the tank around bin 660 ($\approx 5.5\,\mu s$). **(b)** The same signal but filtered and downsampled to emulate UB electronics.

1.1.5. Compatibility mode

Altough the triggers discussed in the previous subsections are meant to function completely autonomously in the SD field, their implementation requires some prior knowledge of the signal one desires to detect. For their use in the Auger observatory, several hyperparameters such as the thresholds of the Th-Trigger, or the window size of the ToT-trigger have been determined in studies ([4], [2], [5]).

These studies were conducted using the predecessor, the **U**nified **B**oard (UB), of the hardware that is being installed during the AugerPrime upgrade of the observatory. Most importantly, the **U**pgraded **U**nified **B**oard (UUB) has a sampling rate that is three times larger (120 MHz) than that of UB electronics (40 MHz). Not only does this raise the number of bins in a standard time trace from 682 to $2^{11} = 2048$, but also drastically reduces the efficiency (in particular for ToT-like triggers) of the above discussed algorithms. Whereas a new FADC bin is measured every 25 ns in a UB station, the triggers would receive a new input every ≈ 8.3 ns in a UUB setting.

The modus operandi elected by the Pierre Auger collaboration to circumvent this problem is to emulate UB electronics using the UUB electronics. This means that measured FADC bins are to be filtered and downsampled before any trigger runs over them. The implementation

he effect this has on measured data is visualized in Figure 1.1.

1.2. Performance

Bibliography

[1] David Nitz. “Surface Detector Trigger Operating Guide”. GAP 2006-057.

[2] Alan Coleman. “The new trigger settings”. GAP 2018-001.

[3] Pierre Billoir. “Peak Searching in FADC traces”. GAP 2002-076.

[4] Xavier Bertou et al. “Calibration of the surface array of the Pierre Auger Observatory”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 568.2 (2006), pp. 839–846.

[5] Pierre Billoir. “Proposition to improve the local trigger of Surface Detector for low energy showers”. GAP 2009-179.

[6] The Pierre Auger Collaboration. *Official Auger Reconstruction and Simulation Framework*. `https://gitlab.iap.kit.edu/auger-observatory/offline`. Cloned on: 13th Dec. 2022 10:00 UTC.

A. Filter and Downsample Algorithm

```python
import numpy as np

def apply_downsampling(pmt : np.ndarray, random_phase : int) -> np.ndarray :
    '''Receive UUB-like ADC trace and filter/downsample to emulate UB electronics'''
    n_bins_uub      = (len(pmt) // 3) * 3               # original trace length
    n_bins_ub       = n_bins_uub // 3                   # downsampled trace length
    sampled_trace   = np.zeros(n_bins_ub)               # downsampled trace container

    # ensure downsampling works as intended
    # cuts away (at most) the last two bins
    if len(pmt) % 3 != 0: pmt = pmt[0 : -(len(pmt) % 3)]

    # see Framework/SDetector/UUBDownsampleFilter.h in Offline main branch for more information
    kFirCoefficients = [ 5, 0, 12, 22, 0, -61, -96, 0, 256, 551, 681, 551, 256, 0, -96, -61, 0, 22, 12, 0, 5 ]
    buffer_length = int(0.5 * len(kFirCoefficients))
    kFirNormalizationBitShift = 11
    kADCsaturation = 4095                               # maximum FADC value: 2^12 - 1

    temp = np.zeros(n_bins_uub + len(kFirCoefficients))

    temp[0 : buffer_length] = pmt[:: -1][-buffer_length - 1 : -1]
    temp[-buffer_length - 1: -1] = pmt[:: -1][0 : buffer_length]
    temp[buffer_length : -buffer_length - 1] = pmt

    # perform downsampling
    for j, coeff in enumerate(kFirCoefficients):
        sampled_trace += [temp[k + j] * coeff for k in range(random_phase, n_bins_uub, 3)]

    # clipping and bitshifting
    sampled_trace = [int(adc) >> kFirNormalizationBitShift for adc in sampled_trace]

    # Simulate saturation of PMTs at 4095 ADC counts ~ 19 VEM LG
    return np.clip(np.array(sampled_trace), a_min = 0, a_max = kADCsaturation)
```

Listing A.1: Python implementation as used in section 1.2 and **??**

```cpp
#ifndef _sdet_UUBDownsampleFilter_h_
#define _sdet_UUBDownsampleFilter_h_

#include <utl/Trace.h>
#include <utl/TimeDistribution.h>
#include <utl/AugerUnits.h>
#include <utl/Math.h>


namespace sdet {

  /**
    Functions to perform the downsample algorithm used for legacy triggers in the UUB firmware

    Besed on descriptions and reference code from Dave Nitz

    The filter will only slightly change the scale of the trace values by factor 2048/2059,
    since the normalization in the firmware uses a bit-shift instead of a division with the
    exact norm. Note that the filter does not produce an UB equivalent trace, this would
    require the division of the trace with an additional factor of 4 which is ommited to
    the preserve dynamic range.

    \author Darko Veberic
    \date 13 Nov 2020
  */

  namespace {

    // FIR coefficients from Dave Nitz's implementation in UUB firmware
    constexpr int kFirCoefficients[] = { 5, 0, 12, 22, 0, -61, -96, 0, 256, 551, 681, 551, 256, 0, -96, -61, 0,
      22, 12, 0, 5 };
    // FIR normalization
    // in firmware an 11-bit right shift is used instead, ie 2048
    constexpr int kFirNormalizationBitShift = 11;
    //const int kFirNormalization = 2059;  // true norm of FIR
    //constexpr double kFirNormalization = (1 << kFirNormalizationBitShift);  // actually used
    constexpr double kUbSampling = 25*utl::nanosecond;
```

```
37      constexpr int kADCSaturation = 4095;
38
39
40      // enforce ADC saturation, both ways
41      inline
42      constexpr
43      int
44      Clip(const int i)
45      {
46        return std::max(0, std::min(i, kADCSaturation));
47      }
48
49    }
50
51    // phase can be one only of { 0, 1, 2 }
52    inline
53    utl::TraceI
54    UUBDownsampleFilter(const utl::TraceI& trace, const int phase = 1)
55    {
56      // input trace is assumed to have 8.333ns binning
57      const int n = trace.GetSize();
58      if (!n)
59        return utl::TraceI(0, kUbSampling);
60      const int m = utl::Length(kFirCoefficients);
61      const int m2 = m / 2;
62      std::vector<int> t;
63      t.reserve(n + 2*m2);
64      // pad front with the first trace values, but backwards
65      for (int i = m2; i; --i)
66        t.push_back(trace[i]);
67      // copy the whole trace
68      for (int i = 0; i < n; ++i)
69        t.push_back(trace[i]);
70      // pad back with the last trace values, but backwards
71      for (int i = 1; i <= m2; ++i)
72        t.push_back(trace[n-1-i]);
73      const int n3 = n / 3;
74      utl::TraceI res(n3, kUbSampling);
75      for (int k = 0; k < n3; ++k) {
76        auto& v = res[k];
77        const int i = 3*k + phase;
78        for (int j = 0; j < m; ++j)
79          v += t[i + j] * kFirCoefficients[j];
80        v >>= kFirNormalizationBitShift;
81        v = Clip(v);
82      }
83      return res;
84    }
85  }
86
87
88  #endif
```

Listing A.2: Implementation in C++, as used in Offline. From [6]